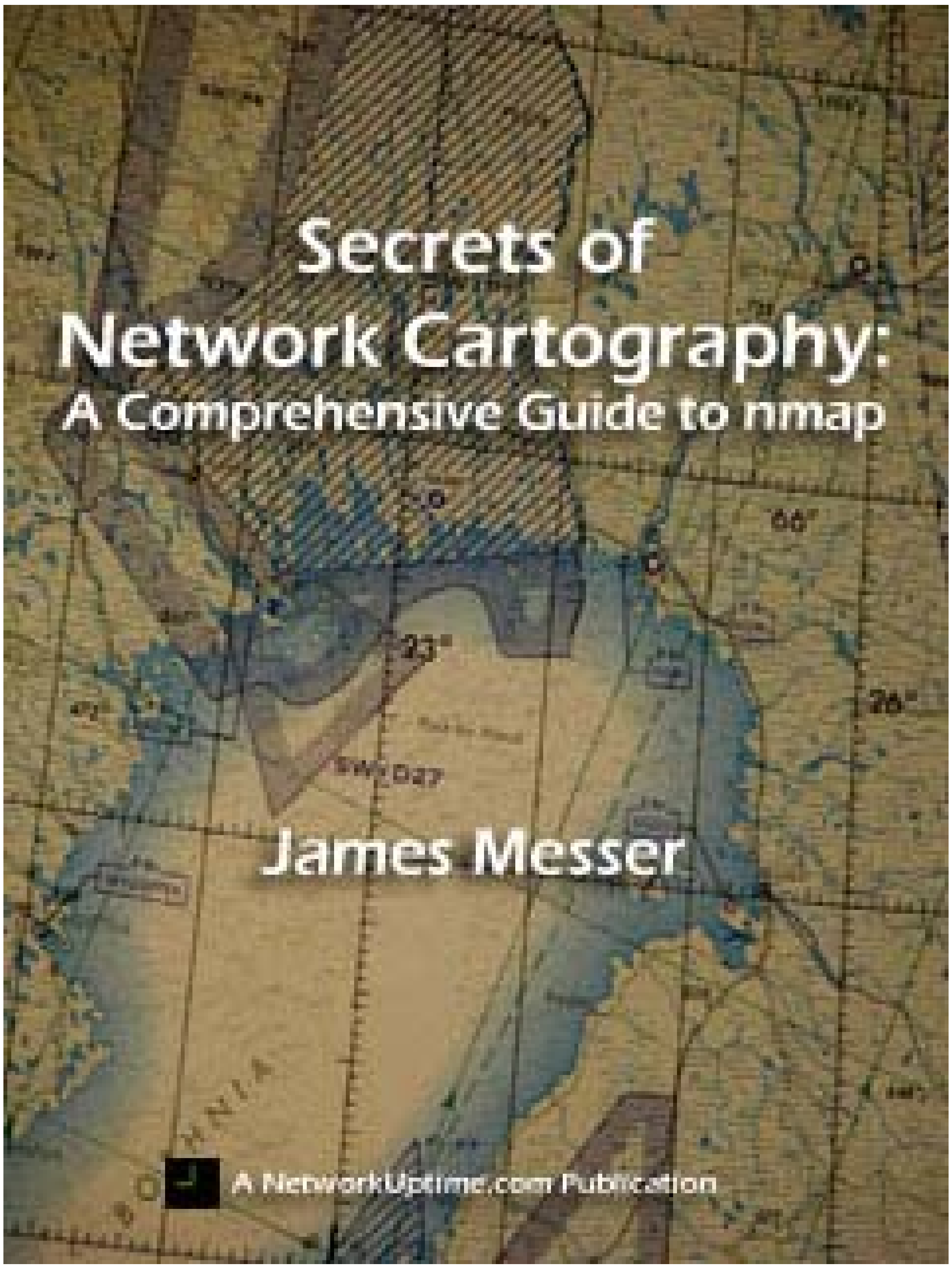

Secrets of
Network Cartography:
A Comprehensive Guide to nmap
James Messer
A NetworkUptime.com Publication

Secrets of Network Cartography:

A Comprehensive Guide to nmap

Written by James Messer

A NetworkUptime.com Publication

Table of Contents

Introduction

Networks are the Wild West of the modern age. The network population is much like that of a frontier town. There's the usual local townsfolk who keep their head down and work amongst themselves, the occasional drifters who come into town and then disappear into the sunset, and there's always at least one black-hat-wearing bad guy who shows up to rob the bank, shoot up the saloon, or just cause a ruckus.

And then, there's the Sheriff. That's you.

In today's modern network, the Sheriff needs more than just a pair of spurs and a six-shooter. Today's network professional requires an eclectic mix of network analyzers, security tools, and multi-functional gadgets. Just like the Wild West, the Sheriff must always stay one step ahead of the bad guys.

Nmap is used every day by thousands of network professionals to keep their systems secure. Nmap's documentation describes itself as a "network exploration tool and security scanner," and it has excelled at these complex capabilities. Nmap tracks down the Wild West town's citizens, identifies each person, and checks them over for potential security gaps. All of these scans are configured, launched, and recorded using nmap's built-in capabilities. With nmap, the Wild West's network becomes a safer and more comfortable place to live.

Nmap is an extremely powerful tool, and one of the most popular security utilities in the open source community. It's written and maintained by "Fyodor" from his web site at http://www.insecure.org/nmap/. The nmap web page is a highly recommended read for its wealth of great security information.

What is nmap?

As its name implies, nmap is a network mapping utility. Provide nmap with a TCP/IP address, and it will identify any open "doors" or ports that might be available on that remote TCP/IP device. The real power behind nmap is the amazing number of scanning techniques and options available! Each nmap scan can be customized to be as blatantly obvious or as invisible as possible. Some nmap scans can forge your identity to make it appear that a separate computer is scanning the network, or simulate multiple scanning decoys on the network! This document will provide an overview of all nmap scanning methods, complete with packet captures and real-world perspectives of how these scans can be best used in enterprise networks.

One powerful aspect of nmap is its ubiquity. Nmap is available on flavors of UNIX, Linux, Windows, and Apple Macintosh OS X. The source code for nmap has been ported to many other operating systems, and it is already included with many UNIX and Linux distributions. You may have nmap already installed on your computer and not even know it!

Nmap runs from the command line of the operating system. This undoubtedly causes discomfort for users who are accustomed to a graphical utility, but understanding the command-line options and syntax is essential for taking advantage of the more advanced functionality that comes from batch files and redirected text. This tutorial will provide a step-by-step approach to understanding the command line, from the simplest options to the most complex.

For those more pictorially inclined, the nmap source distribution includes a graphical front-end for X Window systems called NmapFE. As the term "front-end" suggests, you'll still need to have the nmap binary installed on your system for NmapFE to work properly. Although this tutorial emphasizes the command-line of nmap, it also includes an overview of NmapFE's functionality.

Many Microsoft Windows users may be familiar with NMapWin, an nmap front-end for Windows 2000 and Windows XP. This front-end has not been updated since 2002, and the newest features of nmap are therefore not accessible through NMapWin. Because of these limitations, NMapWin is not included as part of this document. Except for a few Windows-related network shortcomings, nmap still works admirably from the command line on Windows-based computers.

This document will detail every nmap function, display how each option affects network traffic, and show how these functions can be applied to real-world use. Protocol decodes are also included to show how nmap scans will appear when they traverse the network.

Windows Operating Systems and nmap

Although Windows-based operating systems exist on a majority of systems throughout the world, the inner workings of Windows have become somewhat of an irritation for developers of network-related software. Microsoft's implementation of the TCP/IP stack is a bit idiosyncratic, and additional operational restrictions related to Microsoft Windows XP's Service Pack 2 have created ongoing challenges for the developers of these powerful applications.

Many of the most recent issues are based on Microsoft's implementation of raw sockets. Raw sockets are methods built into the operating system that allow a developer to bypass the normal TCP/IP processing of the kernel. This means that programmers can create customized (or raw) TCP/IP frames, a functionality that's critical for security programs. Many of nmap's functions make extensive use of these raw sockets.

With the implementation of Windows XP Service Pack 2 (SP2), Microsoft has removed the ability to create TCP frames through the raw sockets Application Programming Interface (API). UDP packets with spoofed IP addresses are also prevented with SP2. To work around these SP2 raw socket issues, nmap was modified to create raw Ethernet frames instead of raw TCP/IP frames. This fix allows most of the nmap options to work properly, although nmap's raw socket functions can now only create frames on Ethernet networks.

Microsoft also implemented another TCP/IP stack change to Windows XP SP2 that limits the number of simultaneous outbound TCP connections. This has a chilling effect on nmap's TCP connect() scan (-sT), since this scan normally creates many TCP connections. There is at least one non-Microsoft patch that removes this limitation, but the use of this patch is outside the scope of this tutorial. The nmap-hackers mailing list archive has more information on Microsoft's changes and some of the workarounds:

http://seclists.org/lists/nmap-hackers/2004/Jul-Sep/0003.html

These stack changes were also part of Microsoft patch MS05-019 relating to Microsoft Knowledgebase article KB893066, "Vulnerabilities in TCP/IP Could Allow Remote Code Execution and Denial of Service." Even if a Windows XP system hasn't installed Service Pack 2 but still performed the normal security updates, it may exhibit these problems with raw sockets because of MS05-019. It seems clear that these stack revisions may change without notice, and it's highly recommended to stay tuned to the nmap-hackers mailing list.

Creating powerful security tools for Windows-based systems will continue to be an ongoing challenge, but it appears that the nmap developers have worked through many of the current issues. From past experiences, it appears that Windows-based operating systems may not be the best choice for applications such as nmap. Other operating systems have some significant advantages over Windows-based environments when unfettered network access is required for the most efficient network scanning.

Is nmap Good or Evil?

A tool as powerful as nmap can be a double-edged sword. At least one Internet-related reference to nmap describes it as a "hacking tool," and technology purists who don't consider the word "hack" to be a four letter word would certainly agree with this description. It is unfortunate that the term "hacker" has been used incorrectly so often in mainstream society. These uninformed connotations of hacking have overshadowed the original meaning of the word and altered it in horribly negative ways. True hacking is about the pursuit of technology, not about illegal or inappropriate technology subversions.

Still, the ethical question remains; is nmap a useful utility or a nefarious tool? Nmap is about as far as one can go without actively attempting an electronic break-in, but that isn't necessarily a bad feature. It's one thing to drive by a house and see the door propped open. It's something entirely different to stop your car, walk through the door, and steal the television. This is the critical distinction between nmap and more active tools that can exploit vulnerabilities on networked systems, and it's not a small characteristic. Nmap actively scans devices, but nmap does not perform any malicious activity.

The bad guys are already using nmap for reconnaissance, because a single scan can tell you a lot about the open doors and windows in a computer's house. What the bad guys do once they have this information is why they are called the "bad guys."

The good guys are using nmap to make their network safer. The network management team uses nmap to identify unknown IP addresses that appear in reports or in a network analysis trace. The security team uses nmap to scan the internal network for a specific open port that might help to identify the extent of a spyware infestation. The client desktop team uses nmap to inventory a remote location to ensure that all known systems are identified and properly patched against future security concerns.

Nmap is a powerful tool, and its power brings responsibility. Some of nmap's scanning techniques can disable or disrupt production applications. I've personally crashed previously stable applications with a single nmap scan. Many security managers tend to frown on unauthorized users poking around on their network. If you employ nmap, be sure to use it with the knowledge and permission of the network owners.

As a final ethical assessment, one should examine the real-world results of using nmap. Fyodor's nmap man page states that he's used nmap to scan hundreds of thousands of machines, and he's only received one complaint during that time. If the network management and security world has a problem with nmap, they appear to be uncharacteristically quiet regarding its use. It seems that the industry feels that nmap does far more to promote better security than to harm the network.

About This Book

When I started writing this book, I thought it would be a quick twenty-page tutorial on how to use the most basic nmap functions. As I began writing, I found that nmap's feature-rich functionality began to pull me in. My focus completely changed. My two-week writing plans turned into months of nmap network scans, megabytes of protocol decodes, and a complete immersion in nmap's source code.

This book was written from the perspective of the security team, because it is the security team that is managing some of the largest technological responsibilities that our industry has ever experienced. As the first line of this book affirms, our networks really are a Wild West. The security group is always scrambling for methods to combat these constantly increasing and evolving threats.

Nmap is a tool that has been available to network security professionals for years, but it's surprising how many haven't taken advantage of the most basic nmap functionality. I hope that this book will allow security teams to learn more about this incredibly powerful program and help them become better security professionals.

How This Book is Organized

This book consists of eleven chapters, with each chapter designed as a stand-alone set of related topics. If you are already familiar with the basic nmap scans but you need more information about timing and tuning options, you can skip ahead without missing too much.

Chapter 1: The Basics
It's almost impossible to understand the foundation of nmap without an understanding of the underlying protocols, and it's just as difficult to understand nmap's operation without an overview of its architecture. If you've never touched a network or you are just learning nmap, this is a great place to start.

Chapter 2: Nmap Scanning Techniques
Nmap is about scanning, so the first nmap-specific chapter details every possible scanning method. Every nmap scan type is profiled in technical minutia, all the way down to the packet. Each scan technique also includes a list of advantages, disadvantages, and usage recommendations.

Chapter 3: Nmap's Ping Options
The process that runs prior the actual nmap scan is the lesser-known but exceedingly vital nmap ping process. If nmap can't ping a remote device, all of the work that went into choosing a scanning method and its associated options is completely wasted.

Chapter 4: Operating System Fingerprinting
Nmap's operating system fingerprinting process is often overlooked, so this chapter should turn some heads. Nmap can provide amazing detail about a remote device using only fourteen IP frames that never open an application session or log in. If you've never used nmap to fingerprint an OS, you'll be floored after reading this chapter.

Chapter 5: Host and Port Options
Chapters 5 through 10 categorize the remaining nmap options into six distinct categories. Chapter 5 details the nmap options related to hosts and ports. If an nmap scan requires

that a specific IP address be excluded or UDP port 535 be included, these options will provide a wealth of information.

Chapter 6: Logging Options
The scan is simply the means to an end. The end, or the log file, is where all of the important information will be stored. Nmap includes many different logging options, and this chapter will document all of them!

Chapter 7: Real-Time Information Options
As nmap runs, it can provide real-time packet decode information, internal debugging data, or feedback about the version scanning process. Chapter 7 presents each real-time information option and provides suggestions of when each option might be best applied.

Chapter 8: Tuning and Timing Options
Nmap allows customization of the timing and structure of every packet sent to the network. Chapter 8 documents this advanced feature set and shows how it can be applied to almost any scanning scenario.

Chapter 9: Windows-Only Nmap Options
Nmap can run in many different operating systems, but successfully using nmap in Microsoft Windows can be a challenge. Chapter 9 will document each Windows-based option and describes how these options can be used to troubleshoot nmap in a Windows environment.

Chapter 10: Miscellaneous Options
There are some options that just won't fit anywhere else. Chapter 10 provides a catch-all for nmap's more esoteric (or bland) choices.

Chapter 11: Using Nmap in the "Real World"
This is the chapter that started it all. If a security manager wants to know how nmap could assist with security-related challenges, this chapter should provide some talking points.

Conventions Used in This Book

There are some standard conventions used throughout this book that should assist in identifying important information:

- `Fixed-width fonts` are used to display nmap output or to specify command syntax. These fonts are also displayed when nmap options or filenames are referenced.
- Less-than signs (<) and greater-than signs (>) are used to delineate non-optional syntax. For example, the text `–P <portnumber>` signifies that the `portnumber` variable is required whenever the `–P` option is used.
- Square brackets `[ ]` delineate optional syntax. For example, the text `-PS[portlist]` signifies that the `–PS` option could be used without the portlist variable.

The NetworkUptime.com glow-in-the-dark clock highlights "secrets" about nmap or its functions. These important notes are worth the read!

Chapter 1: The Basics

To understand how nmap works, one must also understand the fundamentals of TCP/IP. Nmap uses TCP/IP protocols to query workstations and the responses are interpreted into useful security information. All of the wonderful information that nmap discovers is related to these intricate conversations between nmap and the remote devices.

All computers using the TCP/IP family of protocols follow standard processes when initiating network conversations. Ideally, these processes would be identical regardless of operating system, software version, or hardware manufacturer. In the networking world, however, not every system works exactly the same way. Although these minor differences would usually be considered problematic, nmap takes advantage of these anomalies to provide additional information about the remote system.

The TCP/IP protocols aren't difficult to understand, but each protocol is unique and has its own set of rules and procedures. Once the basics of these protocols are understood, the fundamental operation of nmap becomes much easier to follow. If you're new to networking, don't skip this section!

Internet Protocol

For data to move across the Internet, each device must have an Internet Protocol (IP) address. At its most basic level, IP is a truck-for-hire that carries data shipments across the roads of the network. IP doesn't care what's in the back of the truck; its only goal is to make sure that the truck and its cargo of data get safely from one side of the network to the other.

Just like a real truck, every IP truck needs a starting address to pick up the shipment and a final destination address where the data will be unloaded. In the real world, we think of these as street addresses. In the TCP/IP world, these addresses are usually represented as four decimal numbers between 0 and 255, such as 192.168.0.1 or 10.155.232.17. Before a station can communicate across the network, the IP address of the destination station must be identified so the IP truck will know where to drive. You wouldn't take a trip without knowing where you were going!

The Internet connects many different networks together with devices called routers. As the IP truck drives through the network, it stops at each router to ask for directions. The IP truck also traverses other devices called firewalls or packet filters. As the IP truck travels along its way, a router or firewall may decide that the IP truck's packet isn't permitted to drive along this particular part of the network. If this happens, the firewall or packet filter will obliterate the IP truck from the network, usually without any message back to the originating station. The truck and all of its data are usually never heard from again. Firewalls consistently drop packets from the network, and this often assists nmap in determining if a port is open, closed, or filtered.

Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP) is the most used IP-based protocol on the Internet. An understanding of TCP will provide insight into the most popular protocol in networking and into the inner workings of nmap's powerful scanning engine.

TCP Ports

To move data across the network, simply knowing the IP address of the end station isn't enough to complete the transfer. The cargo in the IP truck must be loaded on one end, and unloaded on the other. The originating station must also identify who will be unloading the truck on the other end, and add that information to the shipment manifest. Instead of names, networks use numbers called ports to represent the entity that will be loading or unloading the data cargo from the IP packet.

A port is a number between 1 and 65,535, and port number references are usually specific to an application. In many cases, an application's port number is a commonly known port number. For example, nearly every web server on the Internet uses port 80 to receive web traffic. Nmap knows that port 80 is rarely filtered, so it uses port 80 as test port to determine device availability.

A list of well known, registered, and dynamic port numbers is maintained by the Internet Assigned Numbers Authority (IANA) at this location:

http://www.iana.org/assignments/port-numbers

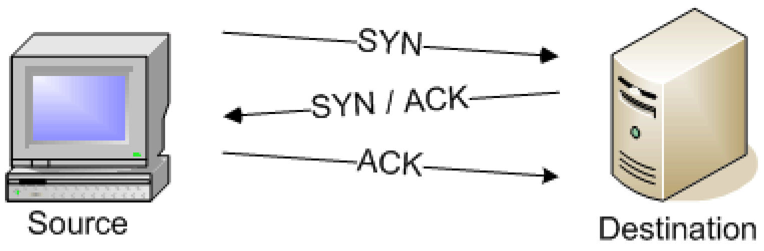
TCP is a very needy protocol. When a frame with TCP data is sent across the network to another station, the sending station must receive an acknowledgement that the data was received properly. If the sending station doesn't receive an acknowledgement after a certain time period, the data is resent in the hopes that it will make it through the second time. This process continues until either the data makes it through, or the transmission process times out.

TCP doesn't need to know how to traverse the network because it relies on IP to get the data to the other side. Once the data makes the trip across the network, TCP takes over and uses its port numbers to determine where to drop the package. It's possible that IP could properly route the data across the network and TCP would try to drop the data at the specified port, but the receiving station may not be listening on that port. The TCP data would have nowhere to go and the entire packet would be discarded.

The TCP Handshake

TCP is not only needy, but it's also a very formal protocol. Before TCP can begin the process of sending data across the network, TCP relies on a "handshake" to set up the conversation between point A and point B. This handshake makes TCP "connection oriented," which means that a connection must be created between the end-stations before TCP communication can proceed.

This handshake is often referred to as the "three way handshake" because of the three frames that pass back and forth:



The First Frame – The initial synchronize (SYN) frame is sent from the station initiating the conversation to the destination station. The SYN frame includes initial sequence numbers and the port that will be used for the conversation, as well as other initialization parameters.

The Second Frame – The destination station receives the SYN frame. If everything is in agreement, it sends an acknowledgement to the SYN (called an ACK) and its own SYN parameters.

The Third Frame – The original station receives the ACK to its original SYN, as well as the SYN from the destination device. Assuming everything is in order, the source station sends an ACK to the destination station's SYN.

This handshake occurs every time a TCP session is established. It's this three-way handshake that allows nmap to gather so much information about the ports on a device.

User Datagram Protocol (UDP)

TCP could be described as a very "needy" protocol. It requires a formal handshaking process, and it demands an acknowledgement for every byte of data that traverses the network.

UDP, however, is the polar opposite. Data that flows across the network with UDP at the helm might get to the other side, or it might not. We don't care! If we really cared enough about the data, we'd send it via TCP so that we'd know it successfully arrived on the other side of the network. It's not that the odds of UDP data getting to the other side are any different than TCP, but UDP doesn't expect a reply. Human beings sometimes interpret that kind of nonchalance as a lack of concern.

In reality, UDP is an extremely valuable protocol! For example, Voice over IP technology uses UDP extensively to send voice information across the network. Since voice information can't be retransmitted, the voice over IP designers realized that UDP is the perfect protocol. If one UDP voice packet is lost that contains a tenth of a second of information, the voice conversation can continue without missing too much of the conversation. Occasionally, the data won't even be missed!

UDP Ports

Just like TCP, UDP uses ports to determine where the data is going once it arrives at the destination. And, just like TCP, UDP ports are numbered between 1 and 65,535. However, TCP and UDP port numbers are unique to their individual protocols. A server that listens for TCP data on port 80 isn't going to accept UDP data on port 80. Most applications use one transport type or the other, but rarely both. If a server shows that port 143 is open, it's important to specify if it's TCP port 143 or UDP port 143. It might be both or it might be none, but it still must be specified.

The Non-existent UDP Handshake

UDP is a connectionless protocol, which means that it doesn't require a formal handshake to get the data flowing. A frame that contains UDP data simply communicates to another station without any prior warning or fanfare. Of course, the receiving station must be configured to receive data on the appropriate UDP port, but no formal handshaking process is required. To send UDP data, it's packaged up in an IP frame and sent on its way – no questions asked!

Internet Control Message Protocol (ICMP)

ICMP is the tattletale of the network. If there's something going on, you can bet that ICMP is going to be there to talk about it. ICMP is a multifaceted protocol that can identify an unreachable destination, redirect traffic to another network, identify routing loops, synchronize clocks, or identify when a router is overloaded.

ICMP is extremely helpful when nmap is scanning for available ports. If some UDP data is sent to another station using a port that's not available, an ICMP "port unreachable" message is usually sent back to the originating station as a notification that data using that particular port number isn't welcome here.

Nmap loves ICMP, because it's so obvious when a port isn't available. It's more difficult when ICMP is filtered or turned off, but nmap can still interpret network responses to determine which ports are open or closed. Since many smart organizations do not allow ICMP to flow through their firewall, it's not always available for nmap to use.

One of ICMP's many functions is to send 'echoes' from one station to another, usually with a program called ping. This functionality is useful when troubleshooting a device's availability. A station sending an ICMP echo request should receive an ICMP echo reply from the other station. By default, nmap sends an ICMP before scanning, although there are scanning options within nmap that can customize this functionality.

The Basics of nmap

Nmap is a powerful utility, but it's somewhat difficult for the novice to understand. There are approximately fifteen different scanning methods within nmap, twenty different options to use when scanning, and the output of nmap can be presented in at least four different ways. Within all of these choices, there are timing variable and packet delay settings that can be tweaked and altered. Although these seem almost overwhelming to the uninitiated, these options have been designed to provide complete customization of the scanning process.

It's easy to understand how to use nmap, but it's more difficult to understand which options to use under which circumstances. After the scan type has been determined, it's equally complex to interpret the results. Each scan type has advantages and disadvantages, and this tutorial will show how every possible scan type can be used for maximum effectiveness.

Most of this tutorial will focus on using nmap in its native form at the command line. The command line requires a bit more typing than a graphical interface, but knowledge of the command line assures that nmap could be used across any platform, operating system, or console. Although a graphical front-end isn't always available, this tutorial will also provide an overview of nmap's front-end capabilities and how they relate to the nmap command-line utility.

The Nmap Scanning Process

Nmap performs four steps during a normal device scan. Some of these steps can be modified or disabled using options on the nmap command line.

1. If a hostname is used as a remote device specification, nmap will perform a DNS lookup prior to the scan. This isn't really an nmap function, but it's useful to mention since this DNS traffic will appear as network traffic and the query will eventually be noted in the DNS logs. If an IP address is used to specify the remote device, this step never occurs. There's no way to disable a DNS lookup when a hostname is specified, unless the hostname and IP address is found in a locally maintained name resolution file such as `hosts` or `lmhosts`.

2. Nmap pings the remote device. This refers to the nmap "ping" process, not (necessarily) a traditional ICMP echo request. Chapter 3 contains more information on nmap's plethora of ping options. This ping process can be disabled with the `–P0` option.

3. If an IP address is specified as the remote device, nmap will perform a reverse DNS lookup in an effort to identify a name that might be associated with the IP address. This is the opposite process of what happens in step 1, where an IP address is found from a hostname specification.

   This process may seem redundant if a DNS lookup is done on step one, but often the results of a name-to-IP-address are different that the results of an IP-address-to-name. Often, the name we use to identify hosts is an alias of the actual host name. For example, if www.microsoft.com is used as a hostname on the nmap command line, the DNS lookup in step one may resolve the IP address as 207.46.19.30. However, a reverse DNS of that IP address in step three might show that address belongs to www.microsoft.com.nsatc.net, a third-party hosting provider for Microsoft.

If this reverse lookup process isn't required or desired, it can be disabled with the `–n` option.

4. Nmap executes the scan. Once the scan is over, this four-step process is completed.

Except for the actual scan process in step four, each of these steps can be disabled or prevented using different IP addressing or nmap options. The nmap process can be as "quiet" or as "loud" as necessary!

If the scan is interrupted (with `CTRL-C`), an "interrupt" process performs a cleanup to close any log files and halt nmap. If the scan is resumed (with the `--resume` option), nmap uses the log file information to begin scanning from the previous location. A normal (`-oN`) or grepable log file (`-oG`) option must be specified to resume the scanning process.

Using nmap from the Command Line

The command line syntax for nmap is similar to any other command line-based utility. Each option is specified one after another on the same line, separated by a space and in no particular order. Nmap uses Unix-style command line syntax by preceding option abbreviations with a single hyphen (`-`) and non-abbreviated options with two hyphens (`--`).

The nmap command

```
nmap -v –p 80 ––randomize_hosts 192.168.0.*
```

will run nmap with the verbose option (`-v`), scan only port 80 (`-p80`), and randomize the selected hosts (`--randomize_hosts`) across the range of 192.168.0.0 through 192.168.0.255. Notice that the abbreviated `–v` and `–p` commands use only one hyphen and the non-abbreviated `––randomize_hosts` option uses two hyphens.

Nmap Target Specifications

Nmap provides many methods of specifying a scan target. The target specification can be placed anywhere on the command line.

A single IP address or hostname can be specified on the command line without any wildcards or lists. For example, the command (`nmap 192.168.1.5` will perform an nmap scan to the 192.168.1.5 address.

A group of hosts can be specified in "slash notation," sometimes referred to as a Classless Inter-Domain Routing (CIDR, pronounced "cider") block notation. The term slash notation refers to the forward-slash that is placed between the IP network address and the number of subnet mask bits. A host specification of 192.168.1.5/24 references a subnet mask of 24 bits, which would scan everything between 192.168.1.0 and 192.168.1.255.

Hyphens, commas, and asterisks can also be used to create a list of hosts. The nmap host specification of 192.168.1-2.* would scan everything between 192.168.1.0 and 192.168.2.255. This could also be specified as 192.168.1,2.0-255, or as 192.168.1-2.1,2-5,6-255. Watch those commas and periods!

The nmap man page throws another twist to the target specification by specifying the networks as the variable values. For example, *.*.1.5 would scan all devices between 1.0.1.5 and 255.255.1.5 (that's a total of 65,535 possible devices!).

Privileged Access
To have access to all possible options, nmap should always be run by a privileged user. A system's privileged users can create custom Ethernet packets that bypass the checks that are normally done by the operating system. With these custom "raw" packets, nmap can manufacture packet header combinations that induce unique responses from the remote stations. These responses then provide nmap with much more information than would normally be available to a non-privileged user.

On a Unix-based system, privileged access means that nmap should run as root, and on Windows-based systems nmap should have Administrator rights. Lack of privileged access doesn't mean that nmap won't work, but certain scanning methods and program options will not be available. In the cases where an option isn't available because of the current permissions, nmap will provide this message:

```
You requested a scan type which requires r00t privileges, and you do
not have them.

QUITTING!
```

Where applicable, the operational details between a privileged and non-privileged user have been detailed in the scan type descriptions of this document.

Nmap Support Files
In addition to the nmap executable, there are six support files that provide nmap with additional information during a scan. These files are `nmap-mac-prefixes`, `nmap-os-fingerprints`, `nmap-protocols`, `nmap-rpc`, `nmap-service-probes`, and `nmap-services`.

Most nmap scans will work properly without these files, but the information provided during the scan will be limited if the files aren't available. Other scans options, such as the operating system fingerprinting option (`-O`), requires the `nmap-os-fingerprints` file or the scan will halt with this error:

```
OS scan requested but I cannot find nmap-os-fingerprints
file. It should be in /usr/local/share/nmap, ~/.nmap, or .
QUITTING!
```

Locating the Support Files
To locate the support files, nmap searches through the file system in this order:

1. A data directory can be specified on the nmap command line with the `--datadir` option. All of the support files should be located in this single directory.

2. An environment variable called `NMAPDIR` can reference the directory containing the support files. This environment variable can be configured in a login profile or using the operating system's `set` command.

3. The real and effective user ID (on POSIX systems) home directory or the location of the nmap executable (on Windows-based systems) is another optional location for the support files. On POSIX systems, the home directory is referenced with a tilde (~), and the files are expected to be in the `~/nmap` directory.

4. The support files can also be stored in the directory where nmap was installed or compiled. This directory is usually `/usr/local/share/nmap` or `/user/share/nmap`.

5. Finally, the current directory is checked for the support files.

Using the Support Files
The information contained in the nmap support files is used differently depending on the selected scan type or option. Not all nmap support files will necessarily be referenced at the same time, but it is possible to create a complex scanning session that would utilize information in all of these support files.

`nmap-mac-prefixes` The `nmap-mac-prefixes` file contains a list of the Organizationally Unique Identifiers (OUIs) assigned to 802-based LAN hardware adapters. Every Ethernet card has a media access control (MAC) address that is burned into the Read Only Memory (ROM) of the network adapter. Many adapter card drivers allow the MAC address to be changed, so this address shouldn't be considered a permanently fixed value. Many network administrators will change their MAC address to something more colorful, such as `11:22:33:44:55:66` or `00:00:BA:5E:BA:11`. This allows the network administrator to easily find their MAC address in a report or log file.

MAC addresses are six bytes long, and are usually referenced in hexadecimal form. They are often written without the traditional "`0x`" reference that usually precedes hex values, and each byte is usually separated with a delimiter such as a hyphen (`-`) or a colon (`:`).

An example of a MAC address assigned to an Ethernet card is `00:11:43:43:A8:34`. The first three bytes, `00:11:43`, are assigned by the Institute of Electrical and Electronics Engineers (IEEE) to the manufacturer of the adapter card. A quick search of the `nmap-mac-prefixes` file shows that the OUI is assigned to Dell.

The last three bytes of this particular MAC address, `43:A8:34`, are considered unique to the adapter card and shouldn't match any other adapter. Although these three bytes provide at least 16,777,215 combinations of unique MAC addresses, adapter card manufacturers have occasionally duplicated MAC addresses due to oversights in the manufacturing process.

The registered list of OUIs is administered and maintained by the IEEE. The OUIs are constantly updated, and a list is available from the IEEE web page.

http://standards.ieee.org/regauth/oui/index.shtml

Additional information on OUIs is available in the IEEE Frequently Asked Questions (FAQs).

http://standards.ieee.org/faqs/OUI.html

The `nmap-mac-prefixes` file is used to correlate and display manufacturer names in the nmap output instead of the raw hexadecimal bytes. This is an excerpt from the `nmap-mac-prefixes` file:

```
-----

0000A3 Network Application Technology
0000A4 Acorn Computers Limited
0000A5 Compatible Systems
0000A6 Network General
0000A7 Network Computing Devices
0000A8 Stratus Computer

-----
```

nmap-os-fingerprints

Although all systems use standards to communicate, every operating system communicates in subtle and unique ways. These differences allow nmap to determine what kind of equipment or what type of operating system is running on a remote device.

The `nmap-os-fingerprints` file is a collection of these unique responses. This collection is referenced during an nmap scan when the operating system fingerprinting (`-O`) option is selected. If nmap scans a device and the responses match a known fingerprint, the name of the device and operating system version will be displayed.

This is the fingerprint of Cisco voice over IP telephone:

```
-----

Fingerprint Cisco 7960 SIP Phone running OS 4.2
Class Cisco | embedded || VoIP phone
TSeq(Class=TD%gcd=<2A004%SI=<28%IPID=I%TS=U)
T1(DF=N%W=3E8%ACK=S++%Flags=AS%Ops=M)
T2(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T3(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T4(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=N)

-----
```

Fyodor has written a comprehensive paper discussing the details of the OS fingerprinting process and the nmap fingerprints. The paper can be viewed here:

http://www.insecure.org/nmap/nmap-fingerprinting-article.html

If a device does not appear in the `nmap-os-fingerprint` file and nmap is able to decisively "read" the fingerprint, a URL will be provided to contribute this new fingerprint to nmap's collection.

nmap-protocols

In the TCP/IP protocol architecture, TCP, UDP, and ICMP are usually described as riding "on top of" IP. IP is the foundation of the communication, and TCP, UDP, and ICMP are three of the protocols that work at a higher layer to accomplish their jobs. In fact, there are over one hundred and thirty different IP-based protocols. Each protocol is assigned a number, and this number is listed in the IP header. TCP is 6, UDP is 17, and ICMP is 1.

The `nmap-protocols` file is used during the IP protocol scan (`-sO`) to assign a known name to any protocols that might be found during the scan. If IP protocol number 8 responds to a scan, the `nmap-protocols` file is referenced:

```
-----

tcp              6     TCP         # Transmission Control
cbt              7     CBT         # CBT
egp              8     EGP         # Exterior Gateway Protocol
igp              9     IGP         # any private interior gateway (used
by Cisco
                                     for their IGRP)
bbn-rcc-mon     10     BBN-RCC-MON # BBN RCC Monitoring
```

```
nvp-ii          11     NVP-II      # Network Voice Protocol
```

IP number 8 refers to Exterior Gateway Protocol (EGP), and nmap will show that EGP was active on this device.

nmap-rpc

Sun's Remote Procedure Call (RPC) architecture was created to provide a client computer with a way to execute procedures on a server. The RPC architecture is available on many different operating systems and platforms.

Using the RPC architecture, each program is assigned a unique hexadecimal number. When a client computer sends data to an RPC server, a program number is used to direct the RPC data to the correct application.

Nmap's RPC scan (`-sR`) will actively search known RPC applications based on the `nmap-rpc` file. Once nmap locates an RPC program, it correlates and displays the program name based on the `nmap-rpc` data. The grinding of this RPC information will automatically run when a version scan (`-sV`) is requested.

This is a sample from the `nmap-rpc` file. The columns correspond to the RPC program name, the hexadecimal RPC program number, and an alias or comment related to the program.

```
rpcbind         100000      portmap sunrpc rpcbind
rstatd          100001      rstat rup perfmeter rstat_svc
rusersd         100002      rusers
nfs             100003      nfsprog nfsd
ypserv          100004      ypprog
mountd          100005      mount showmount
```

-----

nmap-service-probes

The `nmap-service-probes` file is used by the version scan (`-sV`) to determine the application type running on a system (http, ftp, telnet, etc.), the specific application name (Apache httpd, Microsoft IIS, etc.), the version number, and occasionally some additional application information.

This is a sample of an application fingerprint from the `nmap-service-probes` support file:

```
# UW POP2 server on Linux 2.4.18
match pop2 m|^\+ POP2 [-\[\].\w]+ v(20[-.\w]+) server ready\r\n$| v/UW
POP2 server/$1//
```

The `nmap-service-probes` file is nmap-specific, and all of these service signatures have been built over time by a dedicated group of nmap users. The nmap team is always looking for new signatures! Additional information related to version scanning can be found on the nmap website at:

http://www.insecure.org/nmap/versionscan.html

---

nmap-services

Nmap uses the `nmap-services` file to provide additional port detail for almost every scanning method. Every time a port is referenced, it's compared to an available description in this support file. If the `nmap-services` file isn't available, nmap reverts to the `/etc/services` file applicable for the current operating system.

Because the `nmap-services` list is derived from a compilation from many sources, it contains many more records than the Internet Assigned Numbers Authority (IANA) registered port list. Not all of these sources are documented, and many of these port numbers are unique to a single application from a single manufacturer. This list contains information that can apply to almost any network management application! For the latest version of this valuable non-official, non-registered port number list, visit

http://www.graffiti.com/services

This is an excerpt from the `nmap-services` support file:

```
-----

cisco-sccp       2000/tcp callbook sieve # cisco sccp, rfc3028
cisco-sccp       2000/udp callbook     # cisco sccp
dc                   2001/tcp              # or nfr20 web queries
wizard           2001/udp              # curry
globe            2002/tcp
globe            2002/udp
cfingerd         2003/tcp lmtp         # local mail transfer protocol,
gnu finger

-----
```

Chapter 2:

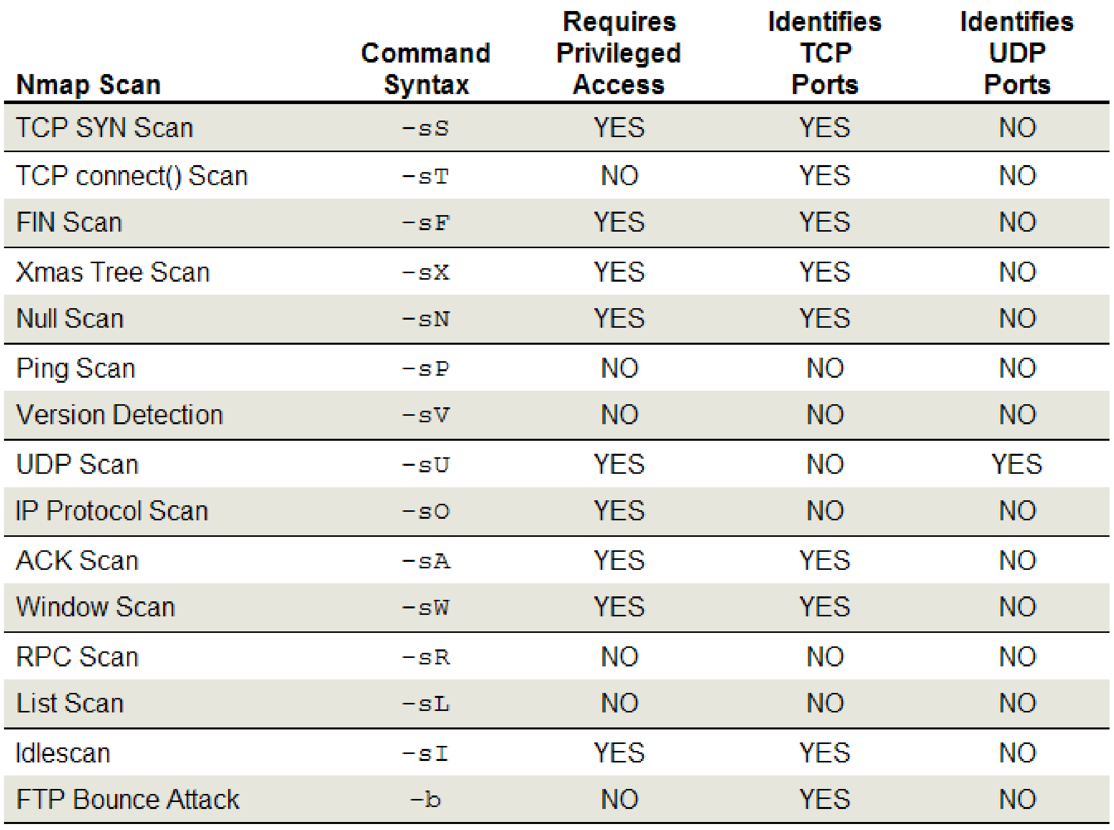
Nmap Scanning Techniques

Nmap includes fifteen separate scanning methods, and each scanning technique has its own characteristics, advantages, and disadvantages. Some of these scanning methods are simple to understand and execute, while others are more complex and require additional information before the scan can begin.

Nearly all of the scans described in this tutorial are demonstrated on an open network without firewalls or packet filters (the ACK scan [`-sA`] is the only scan run through a firewall in the following examples). These scans are described this way for educational purposes, although it could be argued that a more real-world perspective would include descriptions of how these scans operate in all circumstances. Indeed, few modern networks allow all traffic to flow everywhere.

Each scan description includes excerpts from packet captures taken during the scan. In a normal scan, there are hundreds of packets that are transferred between stations. In the operational descriptions shown in this tutorial, most of the packet decode has been removed to focus on the communications that occur between devices for a single port number. The raw packet trace files are available for download if further investigation is required.

Nmap Scan Summary
This chart summarizes the nmap scans and compares the usability for privileged users. The chart also includes a summary of which scans identify TCP ports, and which identify UDP ports.



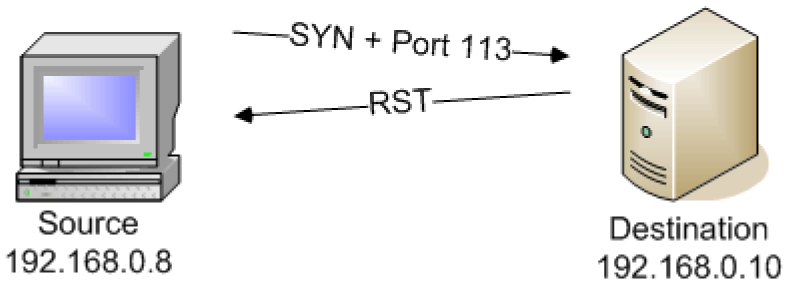
Nmap Scan	Command Syntax	Requires Privileged Access	Identifies TCP Ports	Identifies UDP Ports
TCP SYN Scan	`-sS`	YES	YES	NO
TCP connect() Scan	`-sT`	NO	YES	NO
FIN Scan	`-sF`	YES	YES	NO
Xmas Tree Scan	`-sX`	YES	YES	NO
Null Scan	`-sN`	YES	YES	NO
Ping Scan	`-sP`	NO	NO	NO
Version Detection	`-sV`	NO	NO	NO
UDP Scan	`-sU`	YES	NO	YES
IP Protocol Scan	`-sO`	YES	NO	NO
ACK Scan	`-sA`	YES	YES	NO
Window Scan	`-sW`	YES	YES	NO
RPC Scan	`-sR`	NO	NO	NO
List Scan	`-sL`	NO	NO	NO
Idlescan	`-sI`	YES	YES	NO
FTP Bounce Attack	`-b`	NO	YES	NO

TCP SYN Scan (-sS)
Requires Privileged Access: YES
Identifies TCP Ports: YES
Identifies UDP Ports: NO

The TCP SYN scan uses common methods of port-identification that allow nmap to gather information about open ports without completing the TCP handshake process. When an open port is identified, the TCP handshake is reset before it can be completed. This technique is often referred to as "half open" scanning.

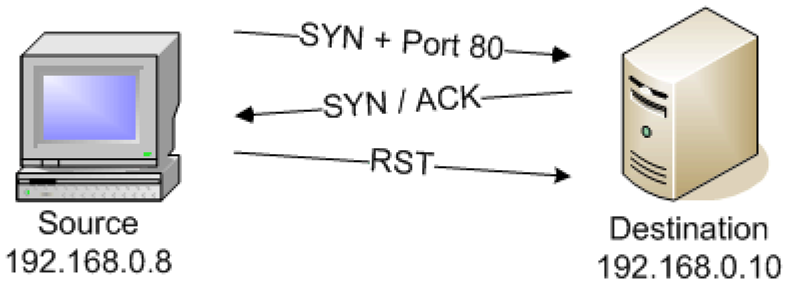
If a scan type is not specified on the nmap command line and nmap currently has privileged access to the host (root or administrator), the TCP SYN scan is used by default.

TCP SYN Scan Operation
Most of the ports queried during the TCP SYN scan will probably be closed. These closed port responses to the TCP SYN frame will be met with a RST frame from the destination station.



```
Source          Destination     Summary
-----------------------------------------------------------------------
[192.168.0.8]   [192.168.0.10]  TCP: D=113 S=57283 SYN SEQ=2360927338
LEN=0 WIN=3072
[192.168.0.10]  [192.168.0.8}   TCP: D=57283 S=113 RST ACK=2360927339
WIN=0
```

If nmap receives an acknowledgment to a SYN request, then the port is open. Nmap then sends an RST to reset the session, and the handshake is never completed.



```
Source          Destination     Summary
-----------------------------------------------------------------------
[192.168.0.8]   [192.168.0.10]  TCP: D=80 S=57283 SYN SEQ=2360927338
LEN=0 WIN=3072
[192.168.0.10]  [192.168.0.8]   TCP: D=57283 S=80 SYN ACK=2360927339
SEQ=1622899389 LEN=0 WIN=65535
[192.168.0.8]   [192.168.0.10]  TCP: D=80 S=57283 RST WIN=0
```

The nmap output shows the results of this TCP SYN scan. As expected, most of the packets sent during this scan were built using the operating system's raw sockets:

```
# nmap -sS -v 192.168.0.10

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-11
12:25 EDT
Initiating SYN Stealth Scan against 192.168.0.10 [1663 ports] at 12:25
Discovered open port 80/tcp on 192.168.0.10
Discovered open port 3389/tcp on 192.168.0.10
Discovered open port 3306/tcp on 192.168.0.10
Discovered open port 139/tcp on 192.168.0.10
Discovered open port 135/tcp on 192.168.0.10
Discovered open port 520/tcp on 192.168.0.10
Discovered open port 445/tcp on 192.168.0.10
The SYN Stealth Scan took 1.35s to scan 1663 total ports.
Host 192.168.0.10 appears to be up ... good.
Interesting ports on 192.168.0.10:
(The 1656 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
80/tcp   open  http
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
520/tcp  open  efs
3306/tcp open  mysql
3389/tcp open  ms-term-serv
MAC Address: 00:30:48:11:AB:5A (Supermicro Computer)

Nmap finished: 1 IP address (1 host up) scanned in 2.117 seconds
               Raw packets sent: 1705 (68.2KB) | Rcvd: 1664 (76.5KB)
#
```

Advantages of the TCP SYN Scan
The TCP SYN scan never actually creates a TCP session, so isn't logged by the destination host's applications. This is a much "quieter" scan than the TCP connect() scan, and there's less visibility in the destination system's application logs since no sessions are ever initiated. Since an application session is never opened, the SYN scan is also less stressful to the application service.

Disadvantages of the TCP SYN Scan
The TCP SYN scan requires that nmap have privileged access to the system. Without privileged access, nmap can't create the raw packets necessary for this half-open connection process.

When to use the TCP SYN Scan
The SYN scan is a common scan when looking for open ports on a remote device, and its simple SYN methodology works on all operating systems. Because it only half-opens the TCP connections, it's considered a very 'clean' scan type.

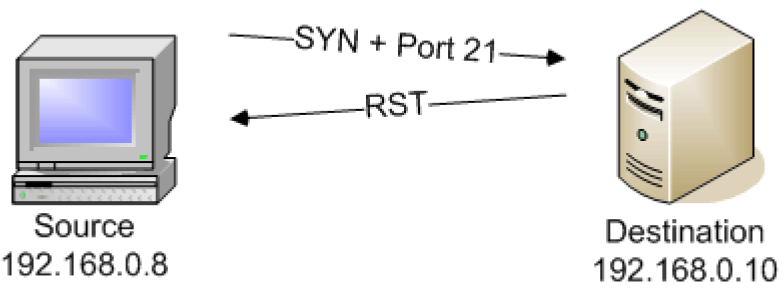
The TCP SYN scan only provides open, closed, or filtered port information. To determine operating system or process version information, more intrusive scanning is required, such as the version scan (`-sV`) or the operating system fingerprinting (`-O`) option.

The TCP SYN scan is the most common scan to use because it works on all networks, across all operating systems, and it's invisible to applications. If the SYN scan didn't work, then TCP wouldn't work!

TCP connect() Scan (`-sT`)
Requires Privileged Access: NO
Identifies TCP Ports: YES
Identifies UDP Ports: NO

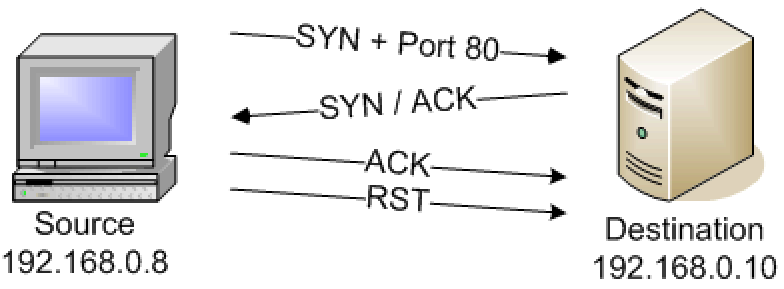
The TCP connect() scan is named after the connect() call that's used by the operating system to initiate a TCP connection to a remote device. Unlike the TCP SYN scan (`-sS`), the TCP connect() scan uses a normal TCP connection to determine if a port is available. This scan method uses the same TCP handshake connection that every other TCP-based application uses on the network.

TCP connect() Scan Operation
The TCP connect() scan to a closed port looks exactly like the TCP SYN scan:



```
Source           Destination     Summary
-----------------------------------------------------------------------
[192.168.0.8]    [192.168.0.10] TCP: D=21 S=41441 SYN SEQ=3365539736
LEN=0 WIN=5840
[192.168.0.10]   [192.168.0.8]   TCP: D=41441 S=21 RST ACK=3365539737
WIN=0
```

A scan to an open port results in a different traffic pattern than the TCP SYN scan:



```
Source           Destination     Summary
-----------------------------------------------------------------------
[192.168.0.8]   [192.168.0.10]   TCP: D=80 S=49389 SYN SEQ=3362197786
LEN=0 WIN=5840
[192.168.0.10] [192.168.0.8]     TCP: D=49389 S=80 SYN ACK=3362197787
SEQ=58695210 LEN=0 WIN=65535
[192.168.0.8]   [192.168.0.10]   TCP: D=80 S=49389 ACK=58695211
WIN<<2=5840
[192.168.0.8]   [192.168.0.10]   TCP: D=80 S=49389 RST ACK=58695211
WIN<<2=5840
```

As the trace file excerpt shows, the TCP connect() scan completed the TCP three-way handshake and then immediately sent a reset (RST) packet to close the connection.

Unlike the TCP SYN scan, the nmap output shows that very few raw packets were required for the TCP connect() process to complete:

```
# nmap -sT -v 192.168.0.10

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-11
12:30 EDT
Initiating Connect() Scan against 192.168.0.10 [1663 ports] at 12:30
Discovered open port 3389/tcp on 192.168.0.10
Discovered open port 80/tcp on 192.168.0.10
Discovered open port 3306/tcp on 192.168.0.10
Discovered open port 445/tcp on 192.168.0.10
Discovered open port 139/tcp on 192.168.0.10
Discovered open port 520/tcp on 192.168.0.10
Discovered open port 135/tcp on 192.168.0.10
The Connect() Scan took 1.45s to scan 1663 total ports.
Host 192.168.0.10 appears to be up ... good.
Interesting ports on 192.168.0.10:
(The 1656 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
80/tcp   open  http
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
520/tcp  open  efs
3306/tcp open  mysql
3389/tcp open  ms-term-serv
MAC Address: 00:30:48:11:AB:5A (Supermicro Computer)

Nmap finished: 1 IP address (1 host up) scanned in 2.242 seconds
               Raw packets sent: 2 (68B) | Rcvd: 1 (46B)
#
```

Advantages of the TCP connect() Scan

No special privileges are required to run the TCP connect() scan. Nmap uses the operating system's normal method of connecting to remote devices via TCP before it tears down the connection with the RST packet. Because these are TCP-based methods that any user can employ, no additional rights or privileges are required.

Disadvantages of the TCP connect() Scan

The disadvantage of this scan is apparent when application connection logs are examined. Since the TCP connect() scan is completing a TCP connection, normal application processes immediately follow. These applications are immediately met with a RST packet, but the application has already provided the appropriate login screen or introductory page. By the time the RST is received, the application initiation process is already well underway and additional system resources are used.

When to use the TCP connect() Scan

Because this scan is so obvious when browsing through the application event logs, it might be considered the TCP scan of last resort. If privileged access isn't available and determination of open TCP ports is absolutely necessary, however, this scan may be the only method available.

The only option to the TCP connect() scan that does not require privileged access but still scans TCP ports is the FTP bounce attack (`-b`). Given the small number of susceptible FTP servers that will participate in a bounce attack, this option is becoming less viable.

I try not to use the connect() scan unless it's absolutely necessary. It's very obvious (in both network traces and in application log files), and it uses many more system and application resources than the SYN scan!

Steath Scanning - The FIN Scan (`-sF`), Xmas Tree Scan (`-sX`), and Null Scan (`-sN`)
Requires Privileged Access: YES
Identifies TCP Ports: YES
Identifies UDP Ports: NO

These three scans are grouped together because their individual functionality is very similar. These are called "stealth" scans because they send a single frame to a TCP port without any TCP handshaking or additional packet transfers. This is a scan type that sends a single frame with the expectation of a single response.

These scans operate by manipulating the bits of the TCP header to induce a response from the remote station. Except for the FIN scan, nmap creates TCP headers that combine bit options that should never occur in the real world. Instead of an obscure bit pattern, the FIN scan creates a scenario that should never occur in the real world. These purposely-mangled TCP header packets are thrown at a remote device, and nmap watches for the responses.

One of the references in RFC 793, Transmission Control Protocol, states that stations receiving information on a closed TCP port should send a RST frame and an available TCP port should not respond at all. During any of these stealth scans, nmap categorizes the responses as either closed, or open|filtered (In computer-speak, a vertical bar between values generally signifies an "or" condition). The open|filtered result is combined because firewalls often drop these frames without a response. Because it's impossible to determine if a missing response was due to an open port or a filtered network connection, there's no way to differentiate between an open port and an administratively dropped frame.

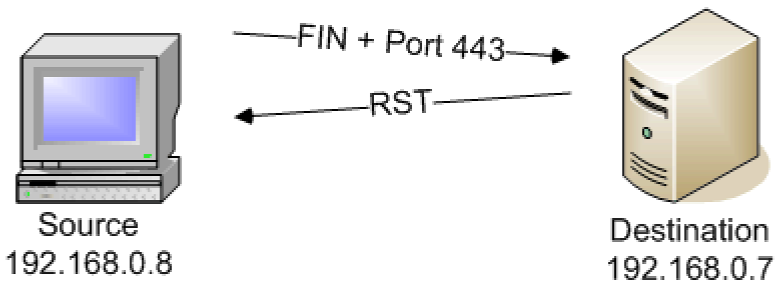
Different TCP/IP stacks may handle these scans in different ways, so it may be necessary to run additional non-stealth scans to get the best overview of responses. For example, Windows-based systems will reply with a RST frame for all queries, regardless of the status of the specific port that was queried. If open|filtered ports appear during an nmap stealth scan, the remote device is definitely not a Windows-based system! Windows-based systems aren't the only TCP/IP stacks that work this way, so special attention should be taken when the results show that all ports are closed; this may not really be the case!

Because these scans create unusual bit combinations in TCP headers, these packets must be built by nmap using the raw sockets functionality of the operating system. Because of these "customized" packets, nmap requires privileged access to perform stealth scans.

FIN, Xmas Tree, and Null Scan Operation
In the following examples, the graphical descriptions and trace files for the open and closed ports will look functionally identical, except that the bits in the TCP flags will be different in each scan type.

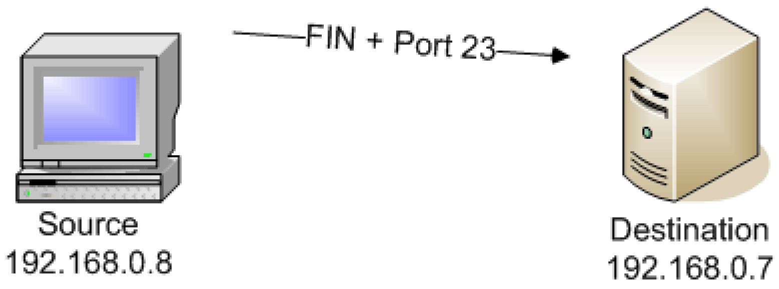
FIN Scan
The FIN scan's "stealth" frames are unusual because they are sent to a device without first going through the normal TCP handshaking. If a TCP session isn't active, the session certainly can't be formally closed!

In this FIN scan, TCP port 443 is closed so the remote station sends a RST frame response to the FIN packet:



```
Source          Destination     Summary
-----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=443 S=62178 FIN SEQ=3532094343 LEN=0
WIN=2048
[192.168.0.7] [192.168.0.8] TCP: D=62178 S=443 RST ACK=3532094343 WIN=0
```

If a port is open on a remote device, no response is received to the FIN scan:



```
Source          Destination     Summary
-----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=23 S=62178 FIN SEQ=3532094343 LEN=0
WIN=2048
```

The nmap output shows the open ports located with the FIN scan:

```
# nmap -sF -v 192.168.0.7

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-23
21:17 EDT
Initiating FIN Scan against 192.168.0.7 [1663 ports] at 21:17
The FIN Scan took 1.51s to scan 1663 total ports.
Host 192.168.0.7 appears to be up ... good.
Interesting ports on 192.168.0.7:
(The 1654 ports scanned but not shown below are in state: closed)
PORT     STATE         SERVICE
21/tcp   open|filtered ftp
22/tcp   open|filtered ssh
23/tcp   open|filtered telnet
79/tcp   open|filtered finger
```

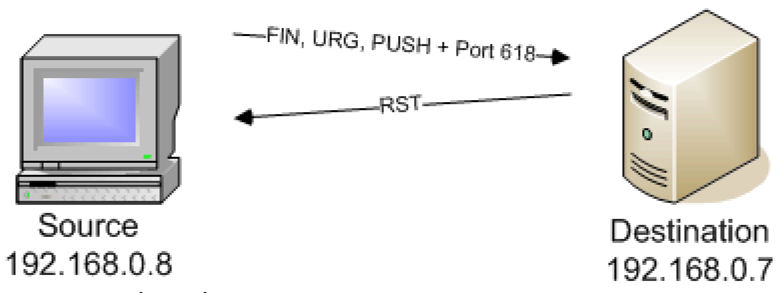
```
110/tcp  open|filtered pop3
111/tcp  open|filtered rpcbind
514/tcp  open|filtered shell
886/tcp  open|filtered unknown
2049/tcp open|filtered nfs
MAC Address: 00:03:47:6D:28:D7 (Intel)

Nmap finished: 1 IP address (1 host up) scanned in 2.276 seconds
               Raw packets sent: 1674 (66.9KB) | Rcvd: 1655 (76.1KB)
#
```

The Xmas Tree Scan (-sX)

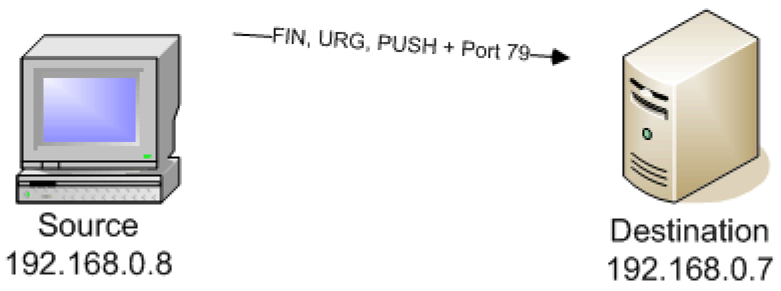
The Xmas tree scan sends a TCP frame to a remote device with the URG, PUSH, and FIN flags set. This is called a Xmas tree scan because of the alternating bits turned on and off in the flags byte (00101001), much like the lights of a Christmas tree.

A closed port responds to a Xmas tree scan with a RST:



```
Source           Destination      Summary
----------------------------------------------------------------------
[192.168.0.8]  [192.168.0.7]  TCP: D=618 S=36793 FIN URG PUSH
SEQ=3378228596 LEN=0 WIN=1024
[192.168.0.7]  [192.168.0.8]  TCP: D=36793 S=618 RST ACK=3378228596
WIN=0
```

Similar to the FIN scan, an open port on a remote station is conspicuous by its silence:



```
Source           Destination     Summary
------------------------------------------------------------------------
[192.168.0.8] [192.168.0.7]  TCP: D=79 S=36793 FIN URG PUSH
SEQ=3378228596 LEN=0 WIN=2048
```

The Xmas tree scan output shows similar results to the FIN scan:

```
# nmap -sX -v 192.168.0.7

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-23
21:18 EDT
Initiating XMAS Scan against 192.168.0.7 [1663 ports] at 21:18
The XMAS Scan took 1.55s to scan 1663 total ports.
Host 192.168.0.7 appears to be up ... good.
Interesting ports on 192.168.0.7:
(The 1654 ports scanned but not shown below are in state: closed)
```

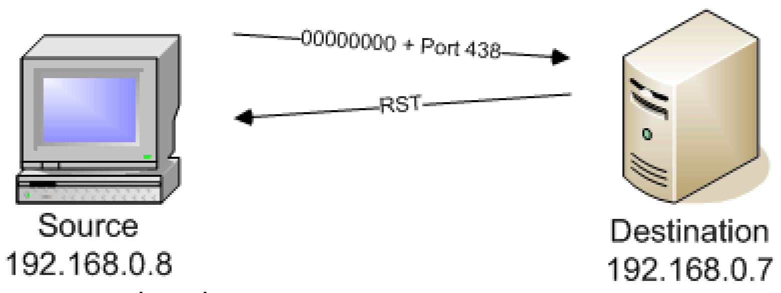
```
PORT     STATE         SERVICE
21/tcp   open|filtered ftp
22/tcp   open|filtered ssh
23/tcp   open|filtered telnet
79/tcp   open|filtered finger
110/tcp  open|filtered pop3
111/tcp  open|filtered rpcbind
514/tcp  open|filtered shell
886/tcp  open|filtered unknown
2049/tcp open|filtered nfs
MAC Address: 00:03:47:6D:28:D7 (Intel)

Nmap finished: 1 IP address (1 host up) scanned in 2.432 seconds
               Raw packets sent: 1674 (66.9KB) | Rcvd: 1655 (76.1KB)
#
```

The Null Scan (-sN)

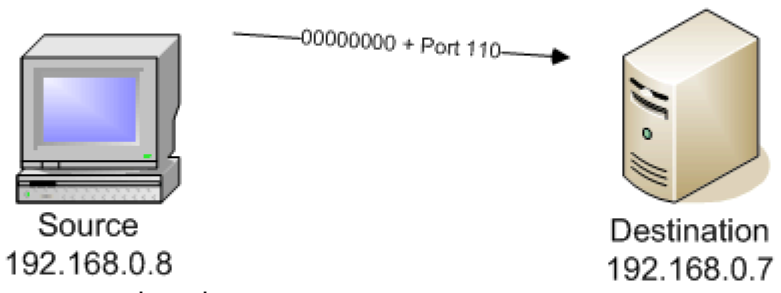
The null scan turns off all flags, creating a lack of TCP flags that should never occur in the real world.

If the port is closed, a RST frame should be returned:



```
Source          Destination      Summary
-----------------------------------------------------------------------
[192.168.0.8]  [192.168.0.7]  TCP: D=438 S=36860     WIN=4096
[192.168.0.7]  [192.168.0.8]  TCP: D=36860 S=438 RST ACK=2135565682
WIN=0
```

As expected, the response of a null scan to an open port results in no response:



```
Source          Destination      Summary
-----------------------------------------------------------------------
[192.168.0.8]  [192.168.0.7]  TCP: D=110 S=36860     WIN=1024
```

The null scan showed the same results as the FIN scan and the Xmas tree scan:

```
# nmap -sN -v 192.168.0.7

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-23
21:19 EDT
Initiating NULL Scan against 192.168.0.7 [1663 ports] at 21:19
The NULL Scan took 1.42s to scan 1663 total ports.
Host 192.168.0.7 appears to be up ... good.
```

```
Interesting ports on 192.168.0.7:
(The 1654 ports scanned but not shown below are in state: closed)
PORT      STATE         SERVICE
21/tcp    open|filtered ftp
22/tcp    open|filtered ssh
23/tcp    open|filtered telnet
79/tcp    open|filtered finger
110/tcp   open|filtered pop3
111/tcp   open|filtered rpcbind
514/tcp   open|filtered shell
886/tcp   open|filtered unknown
2049/tcp  open|filtered nfs
MAC Address: 00:03:47:6D:28:D7 (Intel)

Nmap finished: 1 IP address (1 host up) scanned in 2.251 seconds
               Raw packets sent: 1674 (66.9KB) | Rcvd: 1655 (76.1KB)
#
```

Advantages of the FIN, Xmas Tree, and Null Scan
Since no TCP sessions are created for any of these scans, they are remarkably quiet from the perspective of the remote device's applications. Therefore, none of these scans should appear in any of the application logs.

These scans are also some of the most minimal port-level scans that nmap can execute. For a closed port, only two packets are transferred. A single frame is all that's necessary to find an open port!

Disadvantages of the FIN, Xmas Tree, and Null Scan
Unfortunately, Microsoft's implementation of the TCP/IP stack renders these particular scans less that useful. On a Windows-based computer, all ports will appear to be closed regardless of their actual state. This provides a backhanded advantage, since any device showing open ports must not be a Windows-based device!

These scan types are using packets that do not follow the rules of TCP. To create these specialized packets, the raw sockets capability of the operating system builds the packets from scratch. This avoids the operating system requirements that are usually forced on IP communication, but it also requires that the user running these nmap scans have privileged access to the system.

Some operating systems limit the number of RST packets sent to a single station, and these limits can cause nmap to incorrectly assume that the port is open. I had to disable the ICMP rate limiting on my FreeBSD system to create the examples shown in this document. Disabling rate limiting shouldn't be done in normal use! More information about rate limiting can be found in the FreeBSD FAQ:

http://www.freebsd.org/doc/en_US.ISO8859-1/books/faq/networking.html#ICMP-RESPONSE-BW-LIMIT

When to use the FIN, Xmas Tree, and Null Scan
Although TCP SYN scans are relatively subtle, the FIN, Xmas tree, and null scans are even more invisible on the network. They don't show up in application log files, they take little network bandwidth, and they provide extensive port information on non-Windows

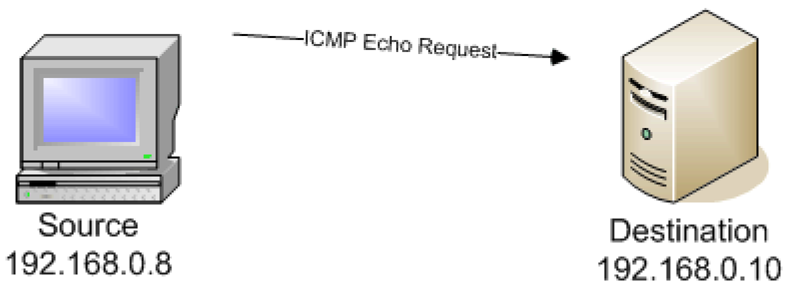
based systems. If the scanned device is susceptible to these odd TCP packets, information can be gathered with only a whisper of network communication!

Ping Scan (-sP)
Requires Privileged Access: NO
Identifies TCP Ports: NO
Identifies UDP Ports: NO

The ping scan is one of the quickest scans that nmap performs, since no actual ports are queried. Unlike a port scan where thousands of packets are transferred between two stations, a ping scan requires only two frames. This scan is useful for locating active devices or determining if ICMP is passing through a firewall.

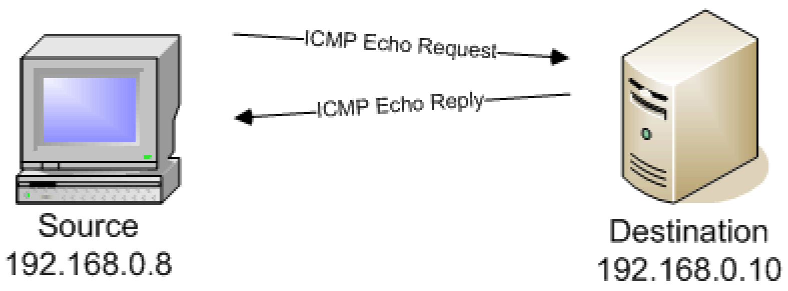
Ping Scan Operation
The ping scan sends a single ICMP echo request from the nmap station to the destination device. A response from an active device will return an ICMP echo reply, unless the IP address is not available on the network or the ICMP protocol is filtered.

If the station isn't available on the network or a packet filter is preventing ICMP packets from passing, there will be no response to the echo frame:



```
Source          Destination     Summary
---------------------------------------------------------------------
[192.168.0.8]  [192.168.0.10] ICMP: Echo
```

A response from an active host will return an ICMP echo reply, unless the IP address is not available on the network or ICMP is filtered.



```
Source          Destination     Summary
---------------------------------------------------------------------
[192.168.0.8]  [192.168.0.10] ICMP: Echo
[192.168.0.10] [192.168.0.8]  ICMP: Echo reply
```

The ping scan output is simple and straightforward:

```
# nmap -sP -v 192.168.0.10
```

```
Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-11
12:40 EDT
Host 192.168.0.10 appears to be up.
MAC Address: 00:30:48:11:AB:5A (Supermicro Computer)
Nmap finished: 1 IP address (1 host up) scanned in 0.778 seconds
               Raw packets sent: 2 (68B) | Rcvd: 1 (46B)
#
```

Advantages of the Ping Scan
The ping scan is one of the most common scanning techniques, and it's used every day by network and server administrators to check for device availability. The ICMP echo request is very innocuous, and these frames are very common on most networks. Unless many IP addresses are chosen for a simultaneous scan, the ICMP echo request will probably not be noticeable.

A ping scan is also very fast, since there are only two frames required to complete the scan to an active workstation. The longest wait time during a ping scan will be the delay that occurs when an unavailable address is scanned.

If the ping scan is successful, then the ICMP protocol is not filtered between the source and destination. Because ICMP can be dangerous in the wrong hands, most security administrators will filter all ICMP packets on their network ingress and egress points.

Disadvantages of the Ping Scan
The ping scan will not interoperate with any other type of scan. If another scan type is specified on the command line with a ping scan, an error message will occur:

```
Sorry, the IPProtoscan, Listscan, and Pingscan (-sO, -sL, -sP) must
currently be used alone rather than combined with other scan types.
QUITTING!
```

The ping scan doesn't provide a lot of information, other than availability and the filtered or unfiltered state of the communication path. Unfortunately, the ping scan can't determine if the lack of an ICMP reply is indicative of an ICMP-filtered connection or an inactive station at that IP address.

When to use the Ping Scan
Ping scans are extremely useful when building an inventory of available stations on a network. This inventory list can then be used to create more complex nmap scans, and some security managers will use these lists to determine if any unknown workstations are active on the network.

Because ping scans only provide uptime information, they are usually performed when more detailed port information is unnecessary. If more detailed information is needed, a different scan type may be a better choice.

ICMP can be used for very evil purposes! If it's not filtered on your firewall, the potential exists for some very malicious activity.

Don't expect the ping scan to work through a firewall. If it does work, the network may be more open than usual on the inside!

Version Detection (`-sV`)
Requires Privileged Access: NO
Identifies TCP Ports: NO
Identifies UDP Ports: NO

Most of nmap's scanning methods are based around the identification of port numbers. However, the version detection scan is most interested in the software applications running on a remote device.

For version detection to work properly, nmap relies on the `nmap-service-probes` file to provide a series of probes and the expected responses. If the `nmap-service-probes` support file is not available, the version detection scan will not run.

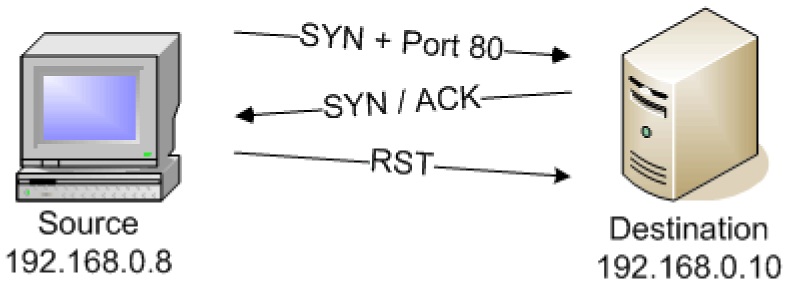
Although other 3rd-party applications have implemented methods of version detection, the process used by the version detection scan is unique to nmap.

USAGE NOTE: On nmap 3.81 using both Linux and Windows, I was able to specify the version detection (`–sV`) option on the command line with the list scan (`-sL`), the ping scan (`-sP`), and the IP protocol scan (`-sO`). Technically speaking, the list scan, ping scan, and IP protocol scan will not run with other scan types on the command line. The version scan parameter was accepted on the command line with these individual scan types, but the version scan did not run and no error message was displayed. To see if the version scan was even active, I removed the `nmap-service-probes` file and received an error message.

Version Detection Operation
The version detection scan runs in conjunction with another scan type that will identify open ports. If another scan type is not specified on the command line, nmap will run a TCP SYN scan (for a privileged user) or a TCP connect() scan (for non-privileged users) prior to running the version detection scan.

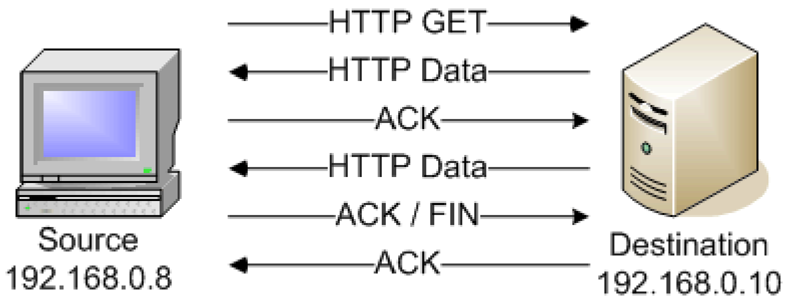
If open ports are found, the version detection scan will begin the probing process with the remote device. The version detection scan communicates directly with the remote application to uncover as much information as possible.

In this example, the default TCP SYN scan runs prior to the version detection scan to identify the open port:



```
Source           Destination      Summary
-----------------------------------------------------------------------
[192.168.0.8] [192.168.0.10] TCP: D=80 S=54490 SYN SEQ=3420003014 LEN=0
WIN=1024
[192.168.0.10] [192.168.0.8] TCP: D=54490 S=80 SYN ACK=3420003015
SEQ=1473373778 LEN=0 WIN=65535
[192.168.0.8] [192.168.0.10] TCP: D=80 S=54490 RST WIN=0
```

After the TCP SYN scan identifies port 80 as open, the version detection process begins. The process shown in this example is for this specific example. Other ports and application will operate differently.



```
Source           Destination      Summary
-----------------------------------------------------------------------
[192.168.0.8] [192.168.0.10] TCP: D=80 S=39330 SYN SEQ=4090323855 LEN=0
WIN=5840
[192.168.0.10] [192.168.0.8] TCP: D=39330 S=80 SYN ACK=4090323856
SEQ=166204426 LEN=0 WIN=65535
[192.168.0.8] [192.168.0.10] TCP: D=80 S=39330 ACK=166204427
WIN<<2=5840
[192.168.0.8] [192.168.0.10] HTTP: C Port=39330 GET / HTTP/1.0
[192.168.0.10] [192.168.0.8] HTTP: R Port=39330 HTTP/1.1 Status=OK-1494
bytes of content \
[192.168.0.8] [192.168.0.10] TCP: D=80 S=39330 ACK=166205875
WIN<<2=8736
[192.168.0.10] [192.168.0.8] HTTP: Continuation of frame 2398;468 Bytes
of data
[192.168.0.8] [192.168.0.10] TCP: D=80 S=39330 FIN ACK=166206344
SEQ=4090323874 LEN=0 WIN<<2=11632
[192.168.0.10] [192.168.0.8] TCP: D=39330 S=80 ACK=4090323875
WIN<<0=65517
```

The scan output displays the application information for each open port, although not all version numbers in this example were identified. The open ports were located by a TCP SYN scan that ran prior to the version scan.

In this example, an open TCP port 520 was located by the SYN scan but the version scan did not recognize the service. A fingerprint was created and nmap provided an URL to use for submission of the unknown service (the URL has been intentionally obfuscated in this document for security reasons).

```
# nmap -sV -v 192.168.0.10
```

```
Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-11
12:41 EDT
Initiating SYN Stealth Scan against 192.168.0.10 [1663 ports] at 12:41
Discovered open port 80/tcp on 192.168.0.10
Discovered open port 3389/tcp on 192.168.0.10
Discovered open port 3306/tcp on 192.168.0.10
Discovered open port 445/tcp on 192.168.0.10
Discovered open port 135/tcp on 192.168.0.10
Discovered open port 139/tcp on 192.168.0.10
Discovered open port 520/tcp on 192.168.0.10
The SYN Stealth Scan took 1.54s to scan 1663 total ports.
Initiating service scan against 7 services on 192.168.0.10 at 12:41
The service scan took 100.01s to scan 7 services on 1 host.
Host 192.168.0.10 appears to be up ... good.
Interesting ports on 192.168.0.10:
(The 1656 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE       VERSION
80/tcp   open  http          Apache httpd 2.0.53 ((Win32))
135/tcp  open  msrpc         Microsoft Windows msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds  Microsoft Windows XP microsoft-ds
520/tcp  open  efs?
3306/tcp open  mysql?
3389/tcp open  microsoft-rdp Microsoft Terminal Service
1 service unrecognized despite returning data. If you know the
service/version, please submit the following fingerprint at
http://www.insecure.org/cgi-bin/xxxxxxxx:
SF-Port520-TCP:V=3.81%D=4/11%Time=425AA8E8%P=i686-pc-linux-
gnu%r(RPCCheck,
SF:24,"\x80\0\0\(r\xfe\x1d\x13\0\0\0\0\0\0\0\x02\0\x01\x86\xa0\x01\x01\
x97
SF:\|\0\0\0\0\0\0\0\0\0\0\0\0");
MAC Address: 00:30:48:11:AB:5A (Supermicro Computer)

Nmap finished: 1 IP address (1 host up) scanned in 102.375 seconds
               Raw packets sent: 1757 (70.3KB) | Rcvd: 1664 (76.5KB)
#
```

The version detection scan can include the `--version_trace` option, which provides a packet-by-packet display of the probing and fingerprinting process. This option is similar to the `--packet_trace` option, except the `--version_trace` option displays only a subset of the frames seen with the `––packet_trace` option.

The version detection scan is one of the scans that runs automatically when the Additional, Advanced, and Aggressive scan (`-A`) is selected. The `–A` option provides an easy way to launch the version detection process in conjunction with other "discovery" scans.

Advantages of Using Version Detection

Version information is valuable information! Version scanning for service information provides easier management of patches and updates. A network security manager can scan every host in an organization to verify that software is at the correct versions. Stations showing older software revisions are identified and further action can be taken.

The version information scan can also assist in locating software that is not complaint with organizational standards. This is also an easy method of verifying the licenses of application services. Nmap can find all of the devices running a specific version of server software to determine if the quantity meets the organization's licensing agreements.

Disadvantages of Using Version Detection
The version scan is very invasive because of the probing that must occur to prompt the service for information. The fingerprint comparison must have information from the application to compare to the fingerprint in the `nmap-service-probes` file, and this process transmits a number of packets between the source and destination.

The version scan also opens sessions with the remote applications, which will often display in an application's log file. These sessions are almost always necessary, and can't be avoided if the version scan is going to decisively determine the application type and version.

Version detection will only work with TCP or UDP port scans. The ping scan (`-sP`), the list scan (`-sL`) and the IP protocol scan (`-sO`) will not run on the same command line with version detection.

The version scan isn't foolproof. The version detection is only as good as the `nmap-service-probe` file, but this support file is constantly under revision. If nmap is unable to match an appropriate fingerprint, check to see if an URL and fingerprint is provided for uploading into the nmap database. Your services can assist in making the next version of nmap even better!

When to use Version Detection
The name and version of a service can provide the security team with information that it can use to keep the network applications patched and up-to-date. The server team can use version scans to confirm that a series of upgrades have been completed successfully. If unknown stations are found during a ping scan, the version scan can help determine what applications these 'rogue' stations are providing!

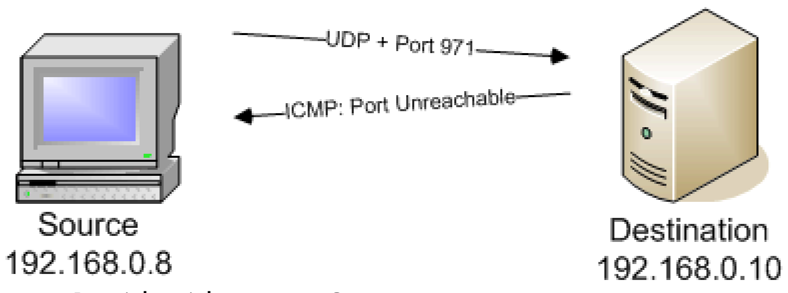
The version detection also shows what other scans might provide when polling network devices for version information. If the security team understands what other people can see, they can revise their security strategies to create a safer computing environment.

UDP Scan (`-sU`)
Requires Privileged Access: YES
Identifies TCP Ports: NO
Identifies UDP Ports: YES

UDP has no need for SYNs, FINs, or any other fancy handshaking. With the UDP protocol, packets are sent and received without warning and prior notice is not usually expected. This lack of a formal communications process greatly simplifies UDP scanning!

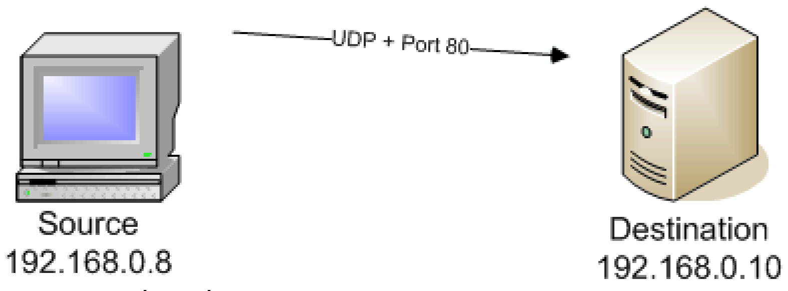
UDP Scan Operation

A station that responds with an ICMP port unreachable is clearly advertising a closed port:



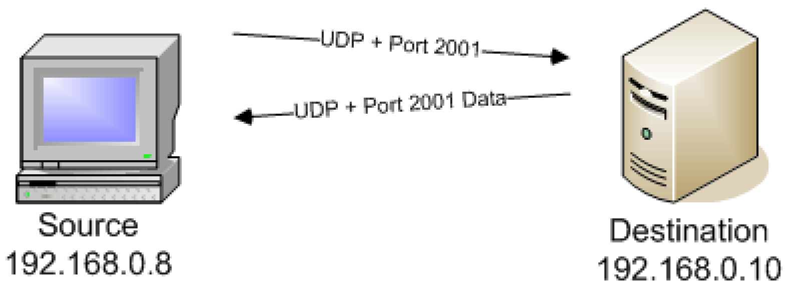
```
Source          Destination      Summary
-----------------------------------------------------------------------
[192.168.0.8]  [192.168.0.10] UDP: D=971 S=43347 LEN=8
[192.168.0.10] [192.168.0.8]  ICMP: Destination unreachable (Port
unreachable)
```

A station that doesn't respond to the UDP scan is considered to be open|filtered:



```
Source          Destination      Summary
-----------------------------------------------------------------------
[192.168.0.8] [192.168.0.10] UDP: D=80 S=43347 LEN=8
```

A station that responds with UDP data is indicative of an open port.



```
Source          Destination      Summary
-----------------------------------------------------------------------
[192.168.0.8]  [192.168.0.10] UDP: D=2001 S=43347 LEN=8
[192.168.0.10] [192.168.0.8]  UDP: D=43347 S=2001 LEN=40
```

The nmap output shows the results of the UDP scan:

```
# nmap -sU -v 192.168.0.10
```

```
Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-11
12:44 EDT
Initiating UDP Scan against 192.168.0.10 [1478 ports] at 12:44
Discovered open port 2001/udp on 192.168.0.10
The UDP Scan took 1.47s to scan 1478 total ports.
Host 192.168.0.10 appears to be up ... good.
Interesting ports on 192.168.0.10:
(The 1468 ports scanned but not shown below are in state: closed)
PORT     STATE         SERVICE
123/udp  open|filtered ntp
137/udp  open|filtered netbios-ns
138/udp  open|filtered netbios-dgm
445/udp  open|filtered microsoft-ds
500/udp  open|filtered isakmp
1031/udp open|filtered iad2
1032/udp open|filtered iad3
1900/udp open|filtered UPnP
2001/udp open          wizard
4500/udp open|filtered sae-urn
MAC Address: 00:30:48:11:AB:5A (Supermicro Computer)

Nmap finished: 1 IP address (1 host up) scanned in 2.241 seconds
               Raw packets sent: 1489 (41.7KB) | Rcvd: 1470 (82.3KB)
#
```

Advantages of the UDP Scan
Since there's no overhead of a TCP handshake, the UDP scan is inherently less "chatty" once it finds an open port. However, if ICMP is responding to each unavailable port, the number of total frames can exceed a TCP scan by about 30%!

Microsoft-based operating systems do not usually implement any type of ICMP rate limiting, so this scan operates very efficiently on Windows-based devices.

Disadvantages of the UDP Scan
The UDP scan only provides port information only. If additional version information is needed, the scan must be supplemented with a version detection scan (`-sV`) or the operating system fingerprinting option (`-O`).

The UDP scan requires privileged access, so this scan option is only available on systems with the appropriate user permissions.

RFC 1812 provides recommendations for limiting the rate of ICMP messages from a single device. The implementation of the rate limiting is not specifically defined, but the RFC suggests methods that would send ICMP messages relative to the number of received frames, ICMP messages every T milliseconds, or ICMP messages relative to the available bandwidth. If a device has implemented ICMP rate limits, nmap will automatically slow the scan rate to match the ICMP rates.

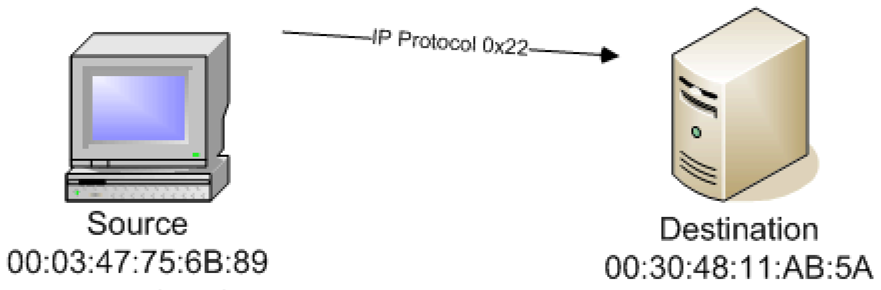
When to use the UDP Scan Because of the huge amount of TCP traffic on most networks, the usefulness of the UDP scan is often incorrectly discounted. There are numerous examples of open UDP ports caused by spyware applications, Trojan horses, and other malicious software. The UDP scan will locate these open ports and provide the security manager with valuable information that can be used to identify and contain these infestations.

IP Protocol Scan (-sO)
Requires Privileged Access: YES
Identifies TCP Ports: NO
Identifies UDP Ports: NO

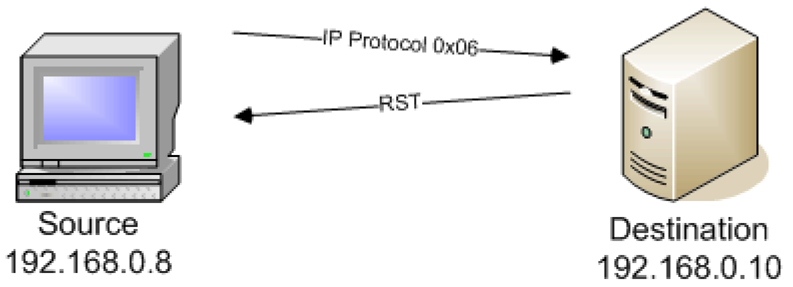
The IP protocol scan is a bit different than the other nmap scans. The IP protocol scan is searching for additional IP protocols in use by the remote station, such as ICMP, TCP, and UDP. If a router is scanned, additional IP protocols such as EGP or IGP may be identified.

The list of IP protocols is found in the `nmap-protocols` file. If the `nmap-protocols` file isn't found, nmap reverts to the `/etc/protocols` file.

IP Protocol Scan Operation
An unavailable IP protocol does not respond to the scan. The MAC addresses are displayed to emphasize the IP layer conversation that occurs between the stations:



```
Source         Destination     Summary
-----------------------------------------------------------------------
Intel 756B89 SprMcr11AB5A  IP: D=[192.168.0.10] S=[192.168.0.8] [0x22:
xns-idp]
```
An available IP Protocol provides a response specific to the protocol type:



```
Source          Destination      Summary
----------------------------------------------------------------------
[192.168.0.8]  [192.168.0.10] TCP: D=44860 S=44860 ACK=637252255
WIN=1024
[192.168.0.10] [192.168.0.8]  TCP: D=44860 S=44860 RST WIN=0
```
The nmap output shows the IP protocol types available on a Windows-based workstation:
```
# nmap -sO -v 192.168.0.10

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-11
12:46 EDT
Initiating IPProto Scan against 192.168.0.10 [256 ports] at 12:46
```

```
Discovered open port 6/ip on 192.168.0.10
Discovered open port 1/ip on 192.168.0.10
The IPProto Scan took 5.70s to scan 256 total ports.
Host 192.168.0.10 appears to be up ... good.
Interesting protocols on 192.168.0.10:
(The 253 protocols scanned but not shown below are in state:
open|filtered)
PROTOCOL STATE    SERVICE
1        open     icmp
6        open     tcp
17       filtered udp
MAC Address: 00:30:48:11:AB:5A (Supermicro Computer)

Nmap finished: 1 IP address (1 host up) scanned in 6.620 seconds
               Raw packets sent: 511 (10.3KB) | Rcvd: 4 (194B)
#
```

Advantages of the IP Protocol Scan
The IP protocol scan locates uncommon IP protocols that may be in use on a system. These are often found on routers and switches that are configured with additional IP protocol support, such as EGP or IGP. Locating these additional protocols can help determine if the destination device is a workstation, a printer, or a router.

Disadvantages of the IP Protocol Scan
When looking at a packet trace, an IP protocol scan looks fairly obvious. Since most networking protocols are based on TCP or UDP, any deviation from those two protocol types is conspicuous. This scan will certainly appear on any network monitoring application that identifies the IP protocol types in use.

This is a sample trace file output from the IP protocol scan. Although the decode summary doesn't show a specific IP protocol value, the source and destination MAC addresses show that this is no ordinary IP conversation.

```
Source        Destination   Summary
-----------------------------------------------------------------------
Intel 756B89 SprMcr11AB5A IP: D=[192.168.0.10] S=[192.168.0.8] LEN=0
ID=9584
Intel 756B89 SprMcr11AB5A IP: D=[192.168.0.10] S=[192.168.0.8] LEN=0
ID=27294
Intel 756B89 SprMcr11AB5A IP: D=[192.168.0.10] S=[192.168.0.8] LEN=0
ID=13528
Intel 756B89 SprMcr11AB5A IP: D=[192.168.0.10] S=[192.168.0.8] LEN=0
ID=36967
Intel 756B89 SprMcr11AB5A IP: D=[192.168.0.10] S=[192.168.0.8] LEN=0
ID=33258
Intel 756B89 SprMcr11AB5A IP: D=[192.168.0.10] S=[192.168.0.8] LEN=0
ID=10186
Intel 756B89 SprMcr11AB5A IP: D=[192.168.0.10] S=[192.168.0.8] LEN=0
ID=43896
```

Since this scan locates IP protocols and doesn't identify open TCP or UDP ports, other scan types will not run in conjunction with the IP protocol scan. If other scan types are specified on the command line with the IP protocol scan, the scan does not run and the following error message is displayed:

```
Sorry, the IPProtoscan, Listscan, and Pingscan (-sO, -sL, -sP) must
currently be used alone rather than combined with other scan types.
QUITTING!
```

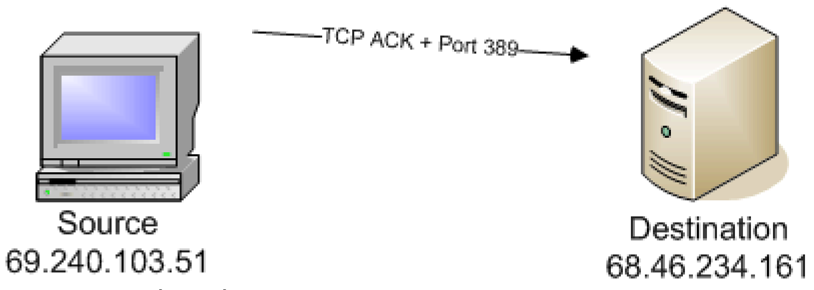
When to use the IP Protocol Scan

The IP protocol scan is useful when looking for "other" protocols that might be in use. If a device is suspected to be a router, the IP protocol scan can locate router protocols that would assist with the identification process.

ACK Scan (`-sA`)
Requires Privileged Access: YES
Identifies TCP Ports: YES
Identifies UDP Ports: NO

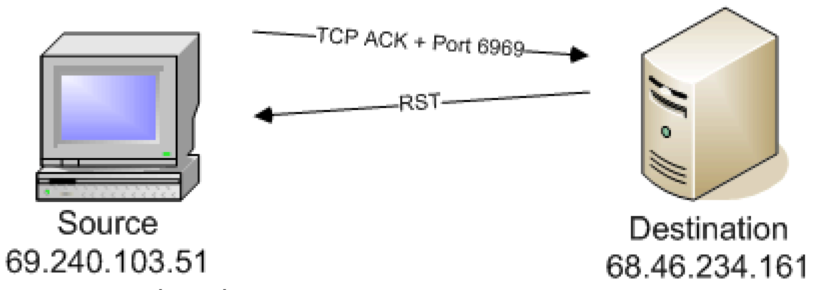
Nmap's unique ACK scan will never locate an open port. The ACK scan only provides a "filtered" or "unfiltered" disposition because it never connects to an application to confirm an "open" state. At face value this appears to be rather limiting, but in reality the ACK scan can characterize the ability of a packet to traverse firewalls or packet filtered links.

ACK Scan Operation
An ACK scan operates by sending a TCP ACK frame to a remote port. If there are no responses or an ICMP destination unreachable message is returned, then the port is considered to be "filtered:"



```
Source          Destination     Summary
-----------------------------------------------------------------------
[69.240.103.51] [68.46.234.161] TCP: D=389 S=38667     ACK=0 WIN=3072
```

If the remote port returns an RST packet, the connection between nmap and the remote device is categorized as unfiltered:



```
Source          Destination     Summary
-----------------------------------------------------------------------
[69.240.103.51] [68.46.234.161] TCP: D=6969 S=38667     ACK=0 WIN=1024
[68.46.234.161] [69.240.103.51] TCP: D=38667 S=6969 RST WIN=0
```

The nmap output shows the scan output through a router, and only a single TCP port was defined as "UNfiltered" (nmap adds the emphasis on the "UN").

```
# nmap -v -sA 68.46.234.161 -P0

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-24
10:40 EDT
```

```
Initiating ACK Scan against pcp05116560pcs.tallah01.fl.comcast.net
(68.46.234.161) [1663 ports] at 10:40
ACK Scan Timing: About 9.02% done; ETC: 10:46 (0:05:03 remaining)
ACK Scan Timing: About 75.68% done; ETC: 10:42 (0:00:36 remaining)
The ACK Scan took 119.13s to scan 1663 total ports.
Host pcp05116560pcs.tallah01.fl.comcast.net (68.46.234.161) appears to
be up ... good.
Interesting ports on pcp05116560pcs.tallah01.fl.comcast.net
(68.46.234.161):
(The 1662 ports scanned but not shown below are in state: filtered)
PORT     STATE      SERVICE
6969/tcp UNfiltered acmsoda

Nmap finished: 1 IP address (1 host up) scanned in 119.271 seconds
               Raw packets sent: 3328 (133KB) | Rcvd: 8 (368B)
#
```

Advantages of the ACK Scan
Since the ACK scan doesn't open any application sessions, the conversation between nmap and the remote device is relatively simple. This scan of a single port is unobtrusive and almost invisible when combined with the other network traffic.

Disadvantages of the ACK Scan
The ACK scan's simplicity is also its largest disadvantage. Because it never tries to connect to a remote device, it can never definitively identify an open port.

When to use the ACK Scan
Although the ACK scan doesn't identify open ports, it does a masterful job of identifying ports that are filtered through a firewall. This list of filtered and unfiltered port numbers is useful as reconnaissance for a more detailed scan that focuses on specific port numbers.

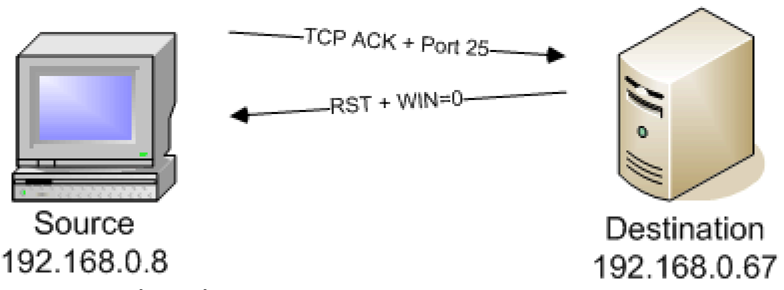
Window Scan (`-sW`)
Requires Privileged Access: YES
Identifies TCP Ports: YES
Identifies UDP Ports: NO

The window scan is similar to an ACK scan, but the window scan has the advantage of identifying open ports. The origins of the window scan can be found in this archive from the nmap-hackers mailing list:

http://seclists.org/lists/nmap-hackers/1999/Jul-Sep/0021.html

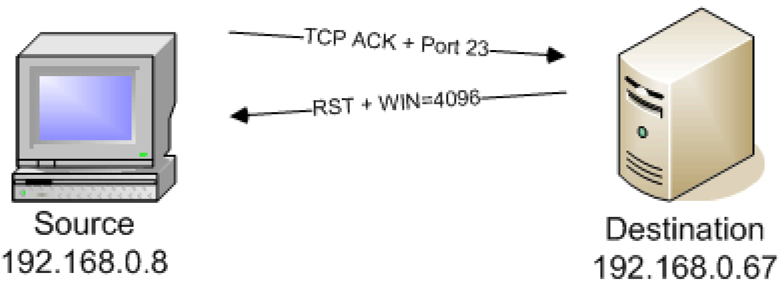
Window Scan Operation
The window scan is named after the TCP sliding window, not the operating system of a similar name. It's called the window scan because some TCP stacks have been found to provide specific window sizes when responding to an RST frame.

A RST frame response from a closed port responds with a window size of zero:



```
Source          Destination     Summary
-----------------------------------------------------------------------
[192.168.0.8]  [192.168.0.67] TCP: D=25 S=62405     ACK=0 WIN=2048
[192.168.0.67] [192.168.0.8]  TCP: D=62405 S=25 RST WIN=0
```

When an open port is sent an ACK frame, the destination station still responds with a RST frame, but the window size is a non-zero value:



```
Source          Destination     Summary
-----------------------------------------------------------------------
[192.168.0.8]   [192.168.0.67]  TCP: D=23 S=62405     ACK=0 WIN=3072
[192.168.0.67]  [192.168.0.8]   TCP: D=62405 S=23 RST WIN=4096
```

The nmap output shows the results of the window scan:

```
# nmap -v -sW 192.168.0.67

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-24
11:37 EDT
Initiating Window Scan against 192.168.0.67 [1663 ports] at 11:37
Discovered open port 23/tcp on 192.168.0.67
Discovered open port 21/tcp on 192.168.0.67
Discovered open port 111/tcp on 192.168.0.67
The Window Scan took 1.91s to scan 1663 total ports.
Host 192.168.0.67 appears to be up ... good.
Interesting ports on 192.168.0.67:
(The 1660 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
21/tcp  open  ftp
23/tcp  open  telnet
111/tcp open  rpcbind
MAC Address: 00:10:A4:07:61:30 (Xircom)

Nmap finished: 1 IP address (1 host up) scanned in 2.749 seconds
               Raw packets sent: 1710 (68.4KB) | Rcvd: 1664 (76.5KB)
#
```

Advantages of the Window Scan

The window scan operates very simply. Nmap sends a single ACK packet request, and a single RST packet is returned for every scanned port. The network traffic is kept to a minimum, and the scan itself looks relatively innocuous when viewed in a protocol decode.

The window scan doesn't open a session, so there's no application log associated with the window scan's method of operation. Unless there are additional firewalls or network limits at the operating system level, the scan should go unnoticed.

Unlike the ACK scan, the window scan is able to identify open ports. If the destination station's operating system is susceptible to this kind of scan, the window scan becomes a very useful method of port identification.

Disadvantages of the Window Scan
The window scan doesn't work on all devices, and the number of operating systems vulnerable to this unintended window size consistency is dwindling as operating systems are upgraded and patched.

The window scan builds custom ACK packets, so privileged access is required to run this scan.

When to use the Window Scan
The window scan is a useful when looking for open ports while simultaneously maintaining a low level of network traffic. When vulnerable operating systems are identified, the window scan provides a low-impact method of locating open ports.

The window scan also works in "reverse." If the window scan is able to identify open ports, then the number of possible operating systems on the remote device can be decreased. To get a list of the operating systems vulnerable to the window scan, refer to the nmap-hackers mailing list archive:

---

http://seclists.org/lists/nmap-hackers/1999/Jul-Sep/0024.html

This archive message shows that referencing the nmap-os-fingerprints file can provide a comprehensive list of known vulnerable operating systems. The command is summarized here:

```
# cat nmap-os-fingerprints | perl -ne 'while(<>) { chomp;if
(/^fingerprint\s+([^\#]+)/i) { if (defined($owin) and defined($cwin)
and $owin ne $cwin)
{ print "$oname ($owin vs. $cwin)\n";}
$oname=$1;undef($cwin);undef($owin);} elsif
(/^T(4|6)\(.*W=([^%]+)/) { if ($1 eq 4){$owin=$2;} else { $cwin = $2;
}}}' | sort –f
```

RPC Scan (`-sR`)
Requires Privileged Access: NO
Identifies TCP Ports: NO
Identifies UDP Ports: NO

A remote program call (RPC) scan is used to locate and identify RPC applications. After open ports are identified with another scan type, the RPC scan sends each open port an RPC null to provoke a response from any RPC application that might be running. The RPC scan runs automatically during a version scan (`-sV`).

The RPC scan is referenced as an RPCGrind scan in the nmap output.

PC Scan Operation

An RPC scan needs a list of open ports before it can begin querying for RPC applications. If a TCP or UDP scan type is not included on the command line, nmap will use the default scan type for the current permissions. For privileged users, nmap performs a TCP SYN scan (`-sS`), and for non-privileged users nmap performs a TCP connect() scan (`-sT`). The examples below display only the RPC scan information, and not the initial port scan.

This trace file shows the RPC query process that occurs for NFS on port 2049. The RPC NULL requests can be clearly seen as PROC=0:

```
Source         Destination    Summary
-----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=2049 S=51008 SYN SEQ=2397105131
LEN=0 WIN=5840
[192.168.0.7] [192.168.0.8] TCP: D=51008 S=2049 SYN ACK=2397105132
SEQ=2432014017 LEN=0 WIN=65535
[192.168.0.8] [192.168.0.7] TCP: D=2049 S=51008 ACK=2432014018
WIN<<2=5840
[192.168.0.8] [192.168.0.7] RPC: C XID=59188766 PROG=Port mapper
VERS=434311 PROC=0(Do nothing)
[192.168.0.7] [192.168.0.8] RPC: R XID=59188766 - Program unavailable
[192.168.0.8] [192.168.0.7] TCP: D=2049 S=51008 ACK=2432014046
WIN<<2=5840
[192.168.0.8] [192.168.0.7] RPC: C XID=59188767 PROG=Remote Statistics
VERS=434311 PROC=0(?)
[192.168.0.7] [192.168.0.8] RPC: R XID=59188767 - Program unavailable
[192.168.0.8] [192.168.0.7] TCP: D=2049 S=51008 ACK=2432014074
WIN<<2=5840
[192.168.0.8] [192.168.0.7] RPC: C XID=59188768 PROG=Remote Users
VERS=434311 PROC=0(?)
[192.168.0.7] [192.168.0.8] RPC: R XID=59188768 - Program unavailable
[192.168.0.8] [192.168.0.7] TCP: D=2049 S=51008 ACK=2432014102
WIN<<2=5840
[192.168.0.8] [192.168.0.7] RPC: C XID=59188769 PROG=NFS VERS=434311
PROC=0(Do nothing)
[192.168.0.7] [192.168.0.8] RPC: R XID=59188769 - Program version
mismatch
[192.168.0.8] [192.168.0.7] TCP: D=2049 S=51008 ACK=2432014138
WIN<<2=5840
[192.168.0.8] [192.168.0.7] TCP: D=2049 S=51008 FIN ACK=2432014138
SEQ=2397105308 LEN=0 WIN<<2=5840
[192.168.0.7] [192.168.0.8] TCP: D=51008 S=2049 ACK=2397105309
WIN<<1=65358
[192.168.0.7] [192.168.0.8] TCP: D=51008 S=2049 FIN ACK=2397105309
SEQ=2432014138 LEN=0 WIN<<1=65358
[192.168.0.8] [192.168.0.7] TCP: D=2049 S=51008 ACK=2432014139
WIN<<2=5840
```

The RPC scan output shown the initial SYN scan (no specific scan type was specified on the command line), and then shows the results of the RPC scan (called RPCGrind Scan in the output):

```
# nmap -v -sR 192.168.0.7

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-22
23:12 EDT
Initiating SYN Stealth Scan against 192.168.0.7 [1663 ports] at 23:12
```

```
Discovered open port 22/tcp on 192.168.0.7
Increasing send delay for 192.168.0.7 from 0 to 5 due to
max_successful_tryno increase to 4
Discovered open port 2049/tcp on 192.168.0.7
Discovered open port 111/tcp on 192.168.0.7
Discovered open port 886/tcp on 192.168.0.7
The SYN Stealth Scan took 10.26s to scan 1663 total ports.
Initiating RPCGrind Scan against 192.168.0.7 at 23:12
The RPCGrind Scan took 1.11s to scan 4 ports on 192.168.0.7.
Host 192.168.0.7 appears to be up ... good.
Interesting ports on 192.168.0.7:
(The 1659 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE                VERSION
22/tcp   open  ssh
111/tcp  open  rpcbind (rpcbind V2-4) 2-4 (rpc #100000)
886/tcp  open  unknown
2049/tcp open  nfs (nfs V2-3)         2-3 (rpc #100003)
MAC Address: 00:03:47:6D:28:D7 (Intel)

Nmap finished: 1 IP address (1 host up) scanned in 12.228 seconds
               Raw packets sent: 1900 (76KB) | Rcvd: 1664 (76.5KB)
#
```

Advantages of the RPC Scan
The RPC scan provides detailed RPC application and version information. If the remote device is running an RPC-based application, nmap will reveal everything!

Even if an RPC application is running on an unexpected port, the RPC scan will find it. The RPC scan sends an RPC null to all open ports, so it's impossible for an RPC application to hide in a little-used port!

Disadvantages of the RPC Scan
The RPC scan opens application sessions, so there will be transaction events in the application logs. Once the application is opened, the size of the conversation will vary. Some open ports will transfer data, but no RPC information. Other RPC applications will send a number of frames as nmap queries for the application name and version number.

Decoys don't currently work in conjunction with an RPC scan, although it may be a future nmap enhancement.

When to use the RPC Scan
If RPC applications are of interest, the RPC scan will efficiently and effectively locate all RPC applications and identify the RPC application name and version. The RPC scan automatically runs during a version scan, combining valuable version information with detailed RPC data.

A UDP scan (-sU) may identify many ports as open|filtered, so the RPC scanning process may take significantly longer if each of these open|filtered ports are checked for RPC applications. Currently, there's no way to tell the RPC scan to ignore open|filtered ports and focus only clearly identified open ports.

List Scan (-sL)
Requires Privileged Access: NO
Identifies TCP Ports: NO
Identifies UDP Ports: NO

The list scan isn't really a scan, but it does provide nmap with some troubleshooting and testing capabilities. The list scan simply lists the IP addresses that would normally be actively scanned.

List Scan Operation
The list scan doesn't ping the host names, and it doesn't send a TCP ACK to the default port number. If reverse DNS resolution is disabled with –n, the list scan is completely silent on the network, with no frames transmitted or received.

By default, nmap will perform a reverse DNS lookup on every IP address in the scan. If the `-n` option isn't specified, this list scan will send traffic over the network and query the DNS server each time an IP address is listed!

The list scan output shows this lack of activity by identifying the IP address as "not scanned:"

```
# nmap -sL -v 192.168.0.10

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-11
13:12 EDT
Host 192.168.0.10 not scanned
Nmap finished: 1 IP address (0 hosts up) scanned in 0.255 seconds
#
```

Advantages of the List Scan
The list scan is a good method to sanity-check a complex nmap scan prior to starting a large batch process or a large group of IP addresses. If any of the IP addresses are defined incorrectly on the command line or the option file, the list scan will identify the errors. These problems can be identified and repaired prior to running any "live" scans.

Disadvantages of the List Scan
The list scan isn't really an active scan. It doesn't show availability, it doesn't find any ports, and it doesn't directly connect with an end device.

The list scan doesn't run at the command line with any other scan type. If another scan type is included on with the list scan, an error message will be displayed:

```
Sorry, the IPProtoscan, Listscan, and Pingscan (-sO, -sL, -sP) must
currently be used alone rather than combined with other scan types.
QUITTING!
```

When to use the List Scan
At first glance, the list scan doesn't appear to be very helpful. What good is a scanning tool that doesn't actually scan anything?

The list scan is often used as a sanity check when a complex scan is defined. If a separate application provides nmap with a list of IP addresses, it may be helpful to have nmap step through a dry-run prior to starting the "production" scan process. For large network audits, this process could be the difference in a successful scan and hours of wasted time!

The list scan is also a useful tool to run "what if" scans. Nmap allows many different IP address options that include randomization, abbreviated subnet mask notation, wildcards, and address ranges. The list scan will confirm the scan process prior to an actual scan.

The list scan implements a reverse DNS lookup for every host specified in the list scan. This means that nmap can be fed a list of IP addresses, and nmap will automatically use the default DNS server to convert the IP addresses to names. This makes it easy to find interesting device names!

If the list scan is being used as a "dry run" prior to a large scan, be sure to use the universal output format (`-oA`) to keep a record of the scan. If something about the scan didn't operate properly, this output can be checked for syntax and errors.

IdleScan (`-sI <zombie host:[probeport]>`)
Requires Privileged Access: YES
Identifies TCP Ports: YES
Identifies UDP Ports: NO

Nmap's idlescan is an ingenious way of scanning a remote device. Nmap uses idlescan to gather port information using another station on the network, and it will appear that the scanning process is initiated from this third-party IP address instead of the nmap station. Although this seems complex, it's a simple process of examining IP fragmentation identification sequences and implementing IP address spoofing.

Idlescan Operation
Before launching an idlescan, a "zombie" station must be identified. This third station will be the pivot point of the idlescan. There are two important requirements associated with this station:

- The unsuspecting zombie station must be idle (thus the name "idlescan"). The idlescan needs an idle zombie workstation to ensure that the IP identification frames will remain consistent throughout the duration of the scan.

- The zombie station must provide consistent and predictable IP identification (IPID) values. If the operating system of the zombie does not provide predictable IPIDs, nmap will provide an warning before the scan begins:

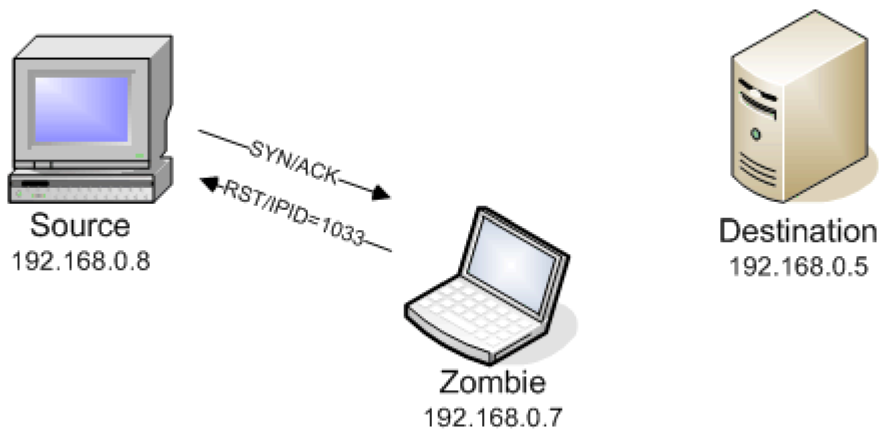
```
      WARNING: Idlescan has erroneously detected phantom ports --
is the proxy
      192.168.0.7 (192.168.0.7) really idle?
```

If the scan cannot complete because of too many inconsistencies, nmap provides a final error message:

```
      Idlescan is unable to obtain meaningful results from proxy
192.168.0.7
      (192.168.0.7).  I'm sorry it didn't work out.
      QUITTING!
```

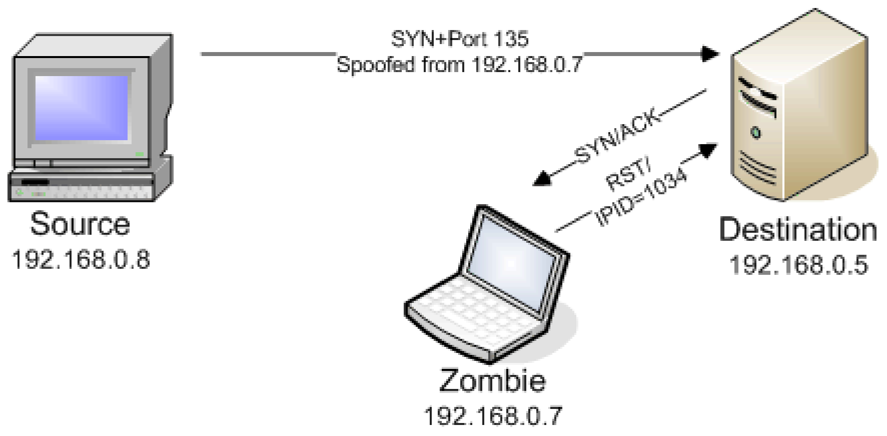
The target of the scan can be any system. These two requires requirements are only associated with zombie station.

To begin the idlescan process, nmap first sends a SYN/ACK to the zombie workstation to induce a RST in return. This RST frame contains the initial IPID that nmap will remember for later.



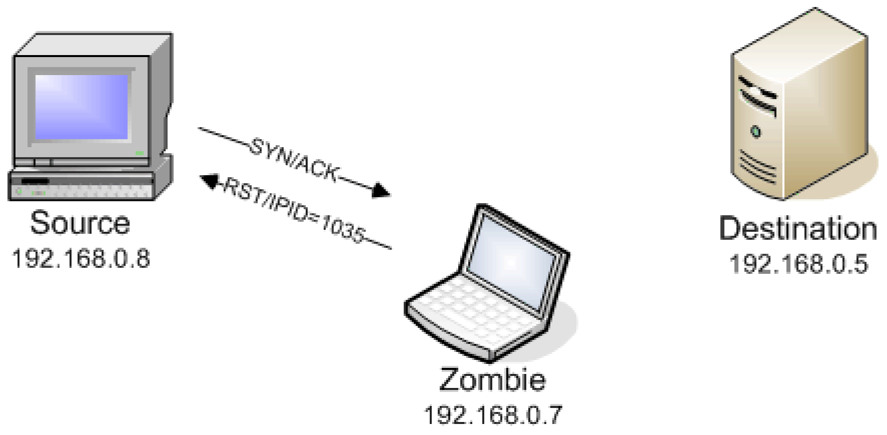
```
Source         Destination    Summary
-----------------------------------------------------------------------
[192.168.0.8]  [192.168.0.7]  IP:  D=[192.168.0.7] S=[192.168.0.8]
LEN=20 ID=26267
[192.168.0.7]  [192.168.0.8]  IP:  D=[192.168.0.8] S=[192.168.0.7]
LEN=20 ID=1033 <--
```

Nmap now sends a SYN frame to the destination address, but nmap spoofs the IP address to make it seem as if the SYN frame was sent from the zombie workstation. If this SYN frame is sent to one of the destination's open ports, the destination address will respond with a SYN/ACK to the previously-spoofed zombie workstation. The zombie workstation won't be expecting the SYN/ACK (after all, it never really sent the SYN), so the zombie will respond to the destination station with a RST. The RST response will, as expected, increment the zombie's IPID.



```
Source         Destination   Summary
----------------------------------------------------------------------
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 SYN SEQ=3699814101 LEN=0
WIN=4096
[192.168.0.5] [192.168.0.7] TCP: D=80 S=135 SYN ACK=3699814102
SEQ=2088485221 LEN=0 WIN=65535
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 RST WIN=0 / IP: ID=1034 <--
```

The final step in the idlescan is for nmap to repeat the original SYN/ACK probe of the zombie station. If the IPID has incremented, then the port that was spoofed in the original SYN frame is open on the destination device. If the IPID has not incremented, then the port is not open.



```
Source         Destination   Summary
-----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] IP:  D=[192.168.0.7] S=[192.168.0.8] LEN=20
ID=11144
[192.168.0.7] [192.168.0.8] IP:  D=[192.168.0.8] S=[192.168.0.7] LEN=20
ID=1035 <--
```

This nmap output shows the results of the idlescan. One of the first messages from nmap is a warning that the nmap station will perform a ping to the destination device unless the `-P0` parameter is specified. If it's important to minimize network visibility, the `-P0` suggestion is a good idea!

```
# nmap -v -sI 192.168.0.7 192.168.0.5
WARNING: Many people use -P0 w/Idlescan to prevent pings from their
true IP.  On the
other hand, timing info Nmap gains from pings can allow for faster,
more
reliable scans.
```

```
Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-24
15:25 EDT
Idlescan using zombie 192.168.0.7 (192.168.0.7:80); Class: Incremental
Initiating Idlescan against 192.168.0.5
Discovered open port 135/tcp on 192.168.0.5
Discovered open port 6969/tcp on 192.168.0.5
Discovered open port 445/tcp on 192.168.0.5
Discovered open port 139/tcp on 192.168.0.5
The Idlescan took 9 seconds to scan 1663 ports.
Host 192.168.0.5 appears to be up ... good.
Interesting ports on 192.168.0.5:
(The 1659 ports scanned but not shown below are in state:
closed|filtered)
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
6969/tcp open  acmsoda
MAC Address: 00:11:43:43:A8:34 (Dell   (WW Pcba Test))

Nmap finished: 1 IP address (1 host up) scanned in 12.629 seconds
               Raw packets sent: 3743 (150KB) | Rcvd: 191 (8786B)
#
```

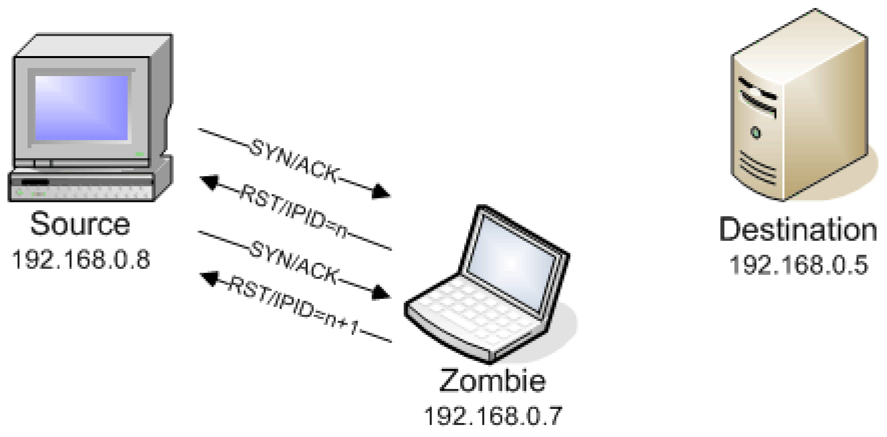
How Idlescan REALLY Works

Although the basic idlescan process is outlined above, nmap's procedure is technically similar but dramatically modified to make the overall idlescan method more efficient. To increase efficiency, nmap handles the spoofing of the SYN frames and the checking of the zombie station's IPID using blocks of port numbers instead of one port at a time.

The explanation that follows is a packet-by-packet description of an idlescan session captured from the network. This process is lengthy, but there are some interesting insights hidden deep within this packet decode. There are a few surprises found in the network trace that aren't visible in the final nmap output!

Idlescan Preparation

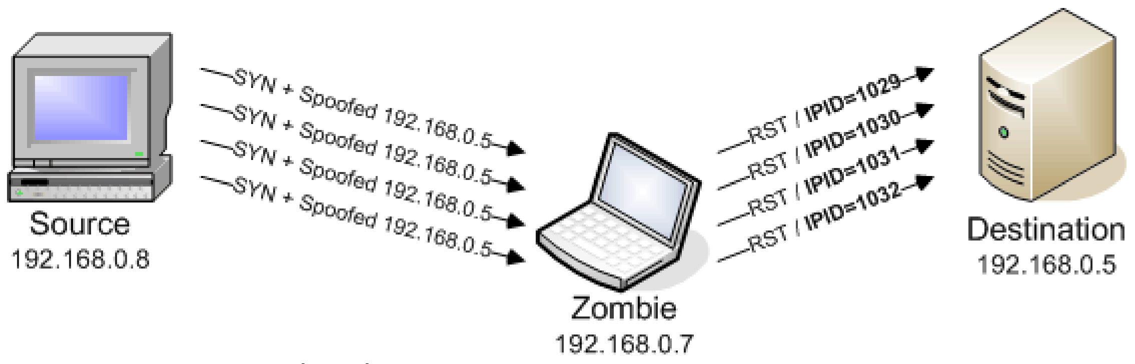
To check a prospective zombie workstation, nmap first sends a series of SYN/ACK frames and watches the RST responses to see if the IPIDs are incrementing consistently. At this point, the idlescan process hasn't yet communicated to the destination station.



The trace file shows the IPID incrementing after a series of six SYN/ACK frames.

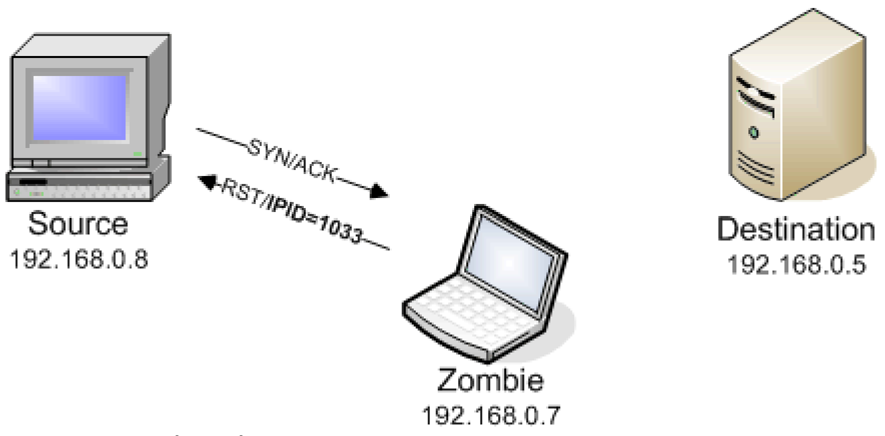
```
Source          Destination     Summary
------------------------------------------------------------------------
[192.168.0.8]  [192.168.0.7]  IP:  D=[192.168.0.7] S=[192.168.0.8]
LEN=20 ID=51592
[192.168.0.7]  [192.168.0.8]  IP:  D=[192.168.0.8] S=[192.168.0.7]
LEN=20 ID=1023 <--
[192.168.0.8]  [192.168.0.7]  IP:  D=[192.168.0.7] S=[192.168.0.8]
LEN=20 ID=12012
[192.168.0.7]  [192.168.0.8]  IP:  D=[192.168.0.8] S=[192.168.0.7]
LEN=20 ID=1024 <--
[192.168.0.8]  [192.168.0.7]  IP:  D=[192.168.0.7] S=[192.168.0.8]
LEN=20 ID=29228
[192.168.0.7]  [192.168.0.8]  IP:  D=[192.168.0.8] S=[192.168.0.7]
LEN=20 ID=1025 <--
[192.168.0.8]  [192.168.0.7]  IP:  D=[192.168.0.7] S=[192.168.0.8]
LEN=20 ID=50056
[192.168.0.7]  [192.168.0.8]  IP:  D=[192.168.0.8] S=[192.168.0.7]
LEN=20 ID=1026 <--
[192.168.0.8]  [192.168.0.7]  IP:  D=[192.168.0.7] S=[192.168.0.8]
LEN=20 ID=36306
[192.168.0.7]  [192.168.0.8]  IP:  D=[192.168.0.8] S=[192.168.0.7]
LEN=20 ID=1027 <--
[192.168.0.8]  [192.168.0.7]  IP:  D=[192.168.0.7] S=[192.168.0.8]
LEN=20 ID=61468
[192.168.0.7]  [192.168.0.8]  IP:  D=[192.168.0.8] S=[192.168.0.7]
LEN=20 ID=1028 <--
```

Having successfully tested source-to-zombie communication, nmap then spoofs the IP address of the destination station and repeats the process four times. The trace file shows the "fake" 192.168.0.5 address communicating to the zombie and the zombie's response to the packet. Notice that the zombie's response goes to the REAL 192.168.0.5 workstation, not the spoofed address. The station that originally spoofed the IP address never sees the resulting frame:



```
Source          Destination     Summary
-----------------------------------------------------------------------
[192.168.0.5]  [192.168.0.7]  IP:  D=[192.168.0.7] S=[192.168.0.5]
LEN=20 ID=3940
[192.168.0.7]  [192.168.0.5]  IP:  D=[192.168.0.5] S=[192.168.0.7]
LEN=20 ID=1029 <--
[192.168.0.5]  [192.168.0.7]  IP:  D=[192.168.0.7] S=[192.168.0.5]
LEN=20 ID=38034
[192.168.0.7]  [192.168.0.5]  IP:  D=[192.168.0.5] S=[192.168.0.7]
LEN=20 ID=1030 <--
[192.168.0.5]  [192.168.0.7]  IP:  D=[192.168.0.7] S=[192.168.0.5]
LEN=20 ID=9069
[192.168.0.7]  [192.168.0.5]  IP:  D=[192.168.0.5] S=[192.168.0.7]
LEN=20 ID=1031 <--
[192.168.0.5]  [192.168.0.7]  IP:  D=[192.168.0.7] S=[192.168.0.5]
LEN=20 ID=30373
[192.168.0.7]  [192.168.0.5]  IP:  D=[192.168.0.5] S=[192.168.0.7]
LEN=20 ID=1032 <--
```

Nmap doesn't know if the responses to these spoofs worked properly because it can't see the response to the spoofed packets. Nmap queries the zombie again for an update on the IPID:



```
Source          Destination     Summary
-----------------------------------------------------------------------
[192.168.0.8]  [192.168.0.7]  IP:  D=[192.168.0.7] S=[192.168.0.8]
LEN=20 ID=26267
[192.168.0.7]  [192.168.0.8]  IP:  D=[192.168.0.8] S=[192.168.0.7]
LEN=20 ID=1033 <--
```

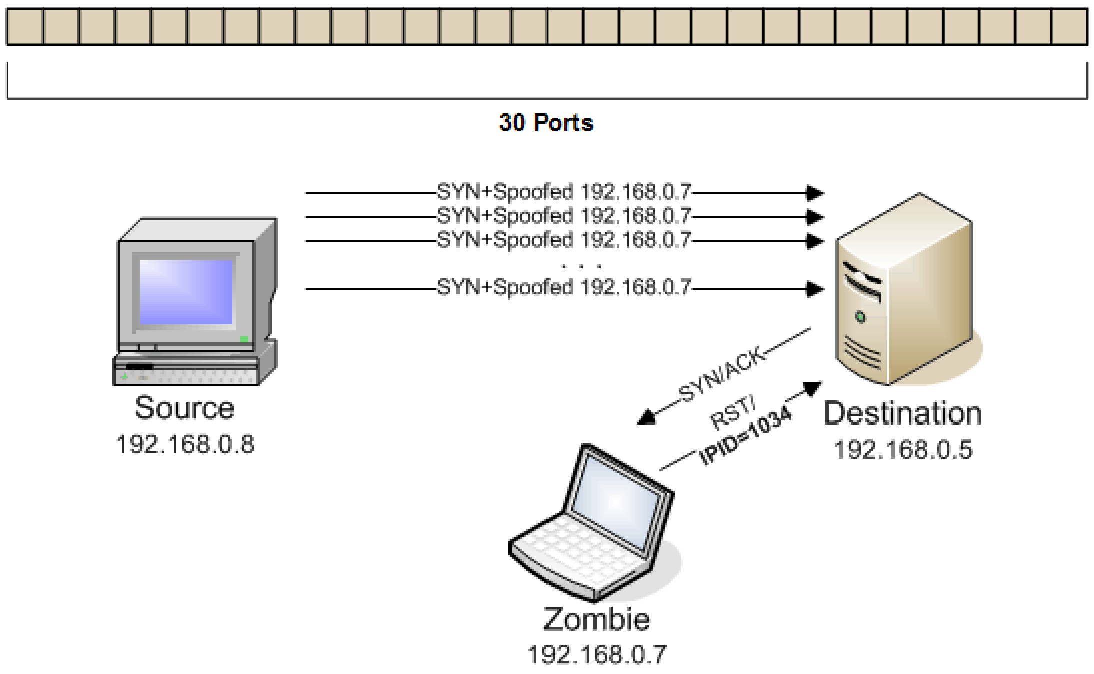
The IPID of 1033 matches the expected number! Now that this IPID analysis process is complete, nmap can begin scanning the destination station.

Deconstructing the Idlescan Process

The basic idlescan operation shown earlier described the scan occurring one port at a time, but nmap knows that most ports will be closed. To improve efficiency, nmap begins scanning multiple ports simultaneously.

Scanning the First Thirty Ports

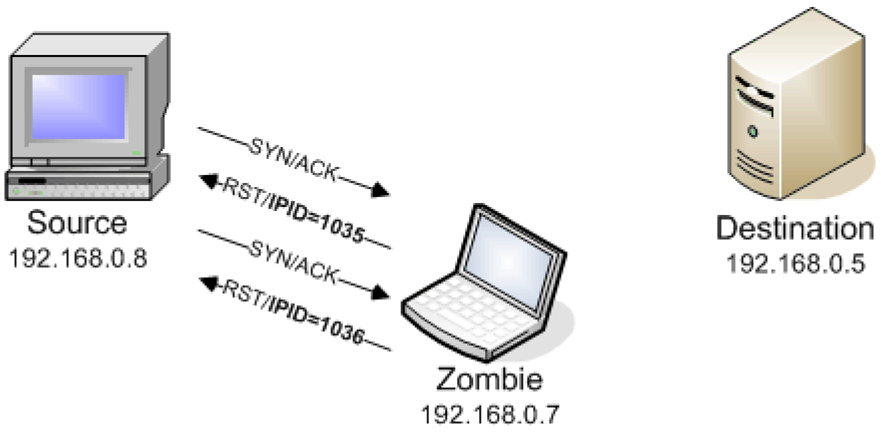
Nmap spoofs the zombie IP address and sends a block of 30 random port numbers to the destination station. To help visualize the idlescan process, this graphic will represent these thirty initially scanned ports:



```
Source          Destination     Summary
--------------------------------------------------------------------
[192.168.0.7] [192.168.0.5] TCP: D=443 S=80 SYN SEQ=3699814101 LEN=0
WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=256 S=80 SYN SEQ=3699814101 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=554 S=80 SYN SEQ=3699814101 LEN=0
WIN=1024
[192.168.0.5] [192.168.0.7] TCP: D=80 S=443 RST ACK=3699814102 WIN=0
[192.168.0.5] 192.168.0.7]  TCP: D=80 S=256 RST ACK=3699814102 WIN=0
[192.168.0.5] 192.168.0.7]  TCP: D=80 S=554 RST ACK=3699814102 WIN=0
[192.168.0.7] [192.168.0.5] TCP: D=389 S=80 SYN SEQ=3699814101 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=3389 S=80 SYN SEQ=3699814101 LEN=0
WIN=3072
[192.168.0.7] [192.168.0.5] TCP: D=80 S=80 SYN SEQ=3699814101 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=53 S=80 SYN SEQ=3699814101 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=25 S=80 SYN SEQ=3699814101 LEN=0
WIN=3072
[192.168.0.7] [192.168.0.5] TCP: D=21 S=80 SYN SEQ=3699814101 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=113 S=80 SYN SEQ=3699814101 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=23 S=80 SYN SEQ=3699814101 LEN=0
WIN=4096
```

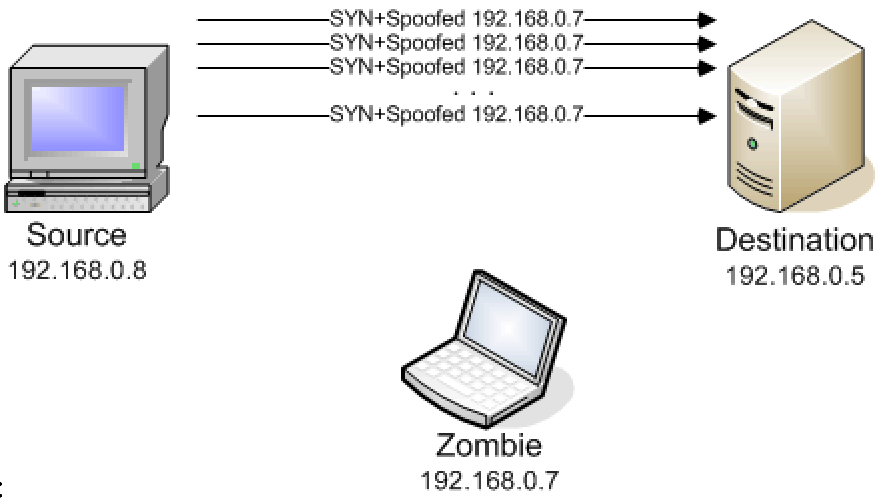
```
[192.168.0.7] [192.168.0.5] TCP: D=636 S=80 SYN SEQ=3699814101 LEN=0
WIN=3072
[192.168.0.7] [192.168.0.5] TCP: D=22 S=80 SYN SEQ=3699814101 LEN=0
WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=1723 S=80 SYN SEQ=3699814101 LEN=0
WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=921 S=80 SYN SEQ=3699814101 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=546 S=80 SYN SEQ=3699814101 LEN=0
WIN=3072
[192.168.0.7] [192.168.0.5] TCP: D=624 S=80 SYN SEQ=3699814101 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=188 S=80 SYN SEQ=3699814101 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=6146 S=80 SYN SEQ=3699814101 LEN=0
WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=7007 S=80 SYN SEQ=3699814101 LEN=0
WIN=3072
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 SYN SEQ=3699814101 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=1436 S=80 SYN SEQ=3699814101 LEN=0
WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=1234 S=80 SYN SEQ=3699814101 LEN=0
WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=808 S=80 SYN SEQ=3699814101 LEN=0
WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=665 S=80 SYN SEQ=3699814101 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=790 S=80 SYN SEQ=3699814101 LEN=0
WIN=3072
[192.168.0.7] [192.168.0.5] TCP: D=873 S=80 SYN SEQ=3699814101 LEN=0
WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=717 S=80 SYN SEQ=3699814101 LEN=0
WIN=3072
[192.168.0.7] [192.168.0.5] TCP: D=185 S=80 SYN SEQ=3699814101 LEN=0
WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=569 S=80 SYN SEQ=3699814101 LEN=0
WIN=3072
[192.168.0.5] [192.168.0.7] TCP: D=80 S=389 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=3389 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=80 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=53 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=25 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=21 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=113 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=23 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=636 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=22 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=1723 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=921 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=546 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=624 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=188 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=6146 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=7007 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=135 SYN ACK=3699814102
SEQ=2088485221 LEN=0 WIN=65535
[192.168.0.5] [192.168.0.7] TCP: D=80 S=1436 RST ACK=3699814102 WIN=0
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 RST WIN=0 / IP: ID=1034 <--
[192.168.0.5] [192.168.0.7] TCP: D=80 S=1234 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=808 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=665 RST ACK=3699814102 WIN=0
```

```
[192.168.0.5] [192.168.0.7] TCP: D=80 S=790 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=873 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=717 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=185 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=569 RST ACK=3699814102 WIN=0
```
Nmap then sends two SYN/ACK frames to the zombie and checks the resulting IPIDs.

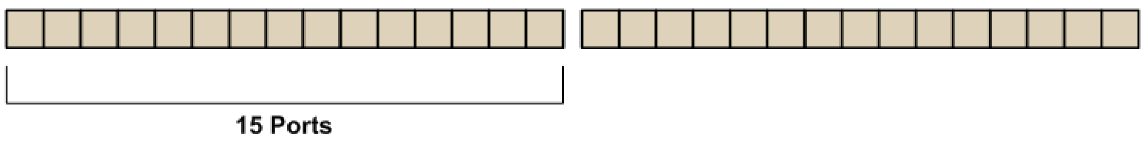


```
Source          Destination     Summary
----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62727 SYN ACK=3093618703
SEQ=717491703 LEN=0 WIN=3072
[192.168.0.7] [192.168.0.8] TCP: D=62727 S=80 RST WIN=0 / IP: ID=1035
<--
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62767 SYN ACK=3093618703
SEQ=717492203 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.8] TCP: D=62767 S=80 RST WIN=0 / IP: ID=1036
<--
```
Nmap can tell from the IPID change that one of the ports sent in that first group of thirty was open!

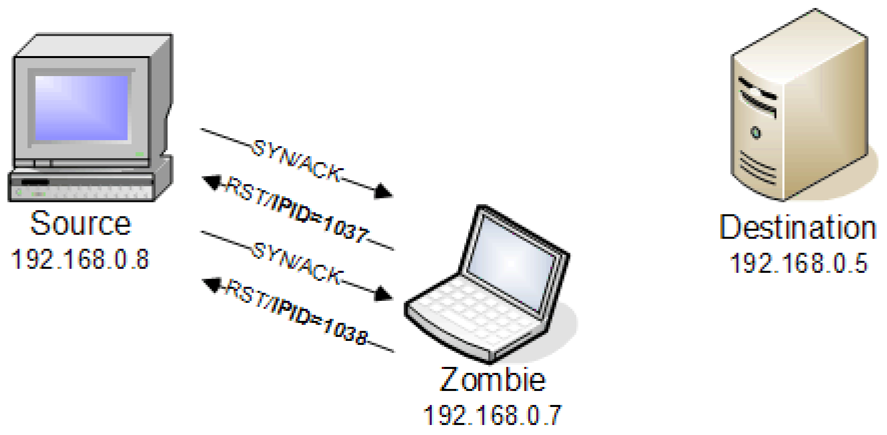
Now that the IPID has changed, nmap rewinds its scan process and begins scanning the same port numbers again, but in smaller groups. Since nmap started with a group of 30, it separates that list in half and begins the scanning process



again:

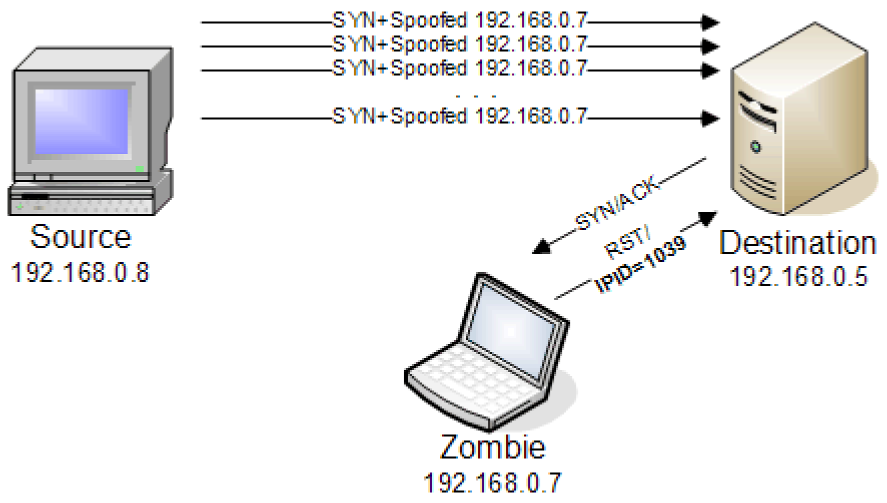


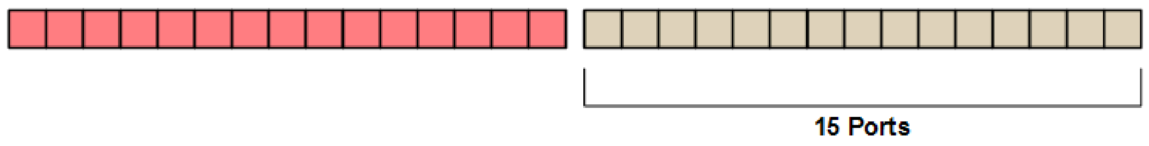
```
Source        Destination   Summary
-----------------------------------------------------------------------
[192.168.0.7] [192.168.0.5] TCP: D=443 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=256 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=554 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=389 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=3072
[192.168.0.5] [192.168.0.7] TCP: D=80 S=443 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=256 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=554 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=389 RST ACK=3699814102 WIN=0
[192.168.0.7] [192.168.0.5] TCP: D=3389 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=80 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=53 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=3072
[192.168.0.7] [192.168.0.5] TCP: D=25 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=3072
[192.168.0.7] [192.168.0.5] TCP: D=21 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=113 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=23 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=636 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=22 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=1723 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=921 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.5] [192.168.0.7] TCP: D=80 S=3389 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=80 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=53 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=25 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=21 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=113 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=23 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=636 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=22 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=1723 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=921 RST ACK=3699814102 WIN=0
```
After these 15 ports are scanned, nmap checks the IPID again:



```
Source         Destination    Summary
-----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62748 SYN ACK=3093618703
SEQ=717492703 LEN=0 WIN=3072
[192.168.0.7] [192.168.0.8] TCP: D=62748 S=80 RST WIN=0 / IP: IPID=1037
<--
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62849 SYN ACK=3093618703
SEQ=717493203 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.8] TCP: D=62849 S=80 RST WIN=0 / IP: IPID=1038
<--
```

The IPID didn't change, so nmap knows that the open port must be one of the remaining 15 ports. However, nmap isn't as efficient as it could be. Instead of splitting apart these remaining 15 ports, it checks all of them!

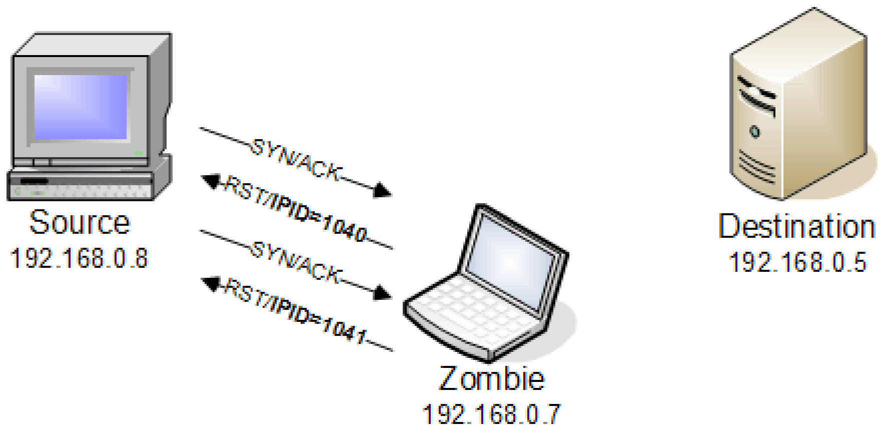




```
Source         Destination    Summary
-----------------------------------------------------------------------
[192.168.0.7] [192.168.0.5] TCP: D=546 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=624 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=3072
[192.168.0.7] [192.168.0.5] TCP: D=188 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=6146 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.5] [192.168.0.7] TCP: D=80 S=546 RST ACK=3699814102 WIN=0
```

```
[192.168.0.5] [192.168.0.7] TCP: D=80 S=624 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=188 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=6146 RST ACK=3699814102 WIN=0
[192.168.0.7] [192.168.0.5] TCP: D=7007 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=1436 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=1234 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=808 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.5] [192.168.0.7] TCP: D=80 S=7007 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=135 SYN ACK=3699814102
SEQ=2088563926 LEN=0 WIN=65535
[192.168.0.5] [192.168.0.7] TCP: D=80 S=1436 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=1234 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=808 RST ACK=3699814102 WIN=0
[192.168.0.7] [192.168.0.5] TCP: D=665 S=80 SYN (Retransmission of
Frame 60) SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=790 S=80 SYN (Retransmission of
Frame 61) SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=873 S=80 SYN (Retransmission of
Frame 62) SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=717 S=80 SYN (Retransmission of
Frame 63) SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=185 S=80 SYN (Retransmission of
Frame 64) SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=569 S=80 SYN (Retransmission of
Frame 65) SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.5] [192.168.0.7] TCP: D=80 S=665 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=790 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=873 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=717 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=185 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=569 RST ACK=3699814102 WIN=0
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 RST WIN=0 / IP: IPID=1039
<--
```

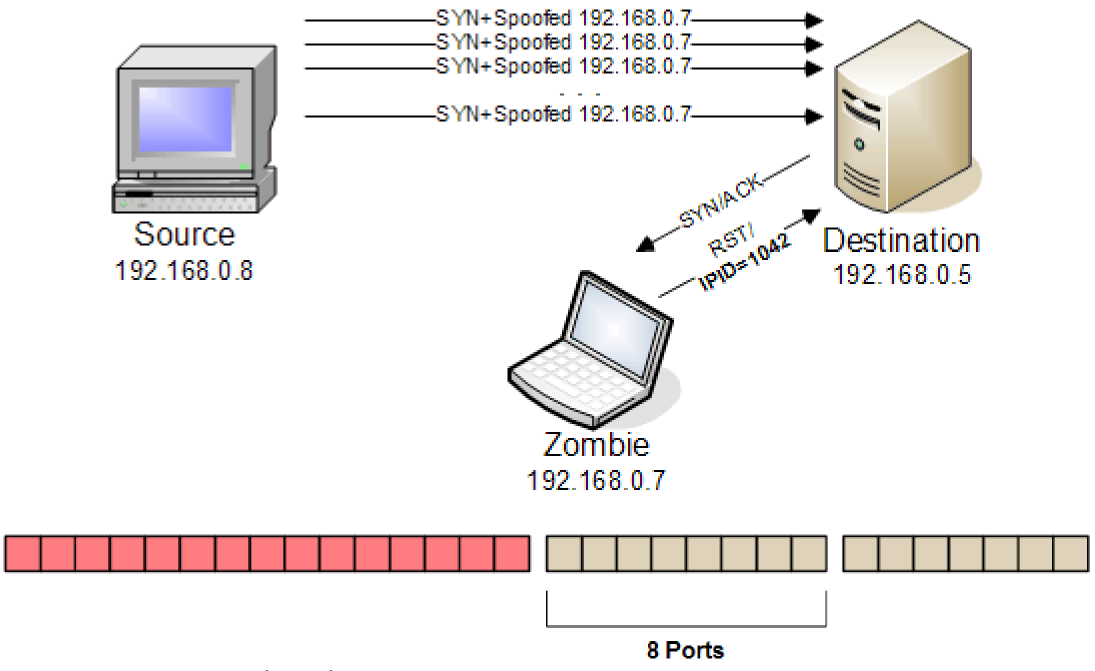
Nmap checks the IPIDs to determine if the open port was in that last batch. Of course, we've already determined it's there:



```
Source        Destination    Summary
----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62803 SYN ACK=3093618703
SEQ=717493703 LEN=0 WIN=4096
```

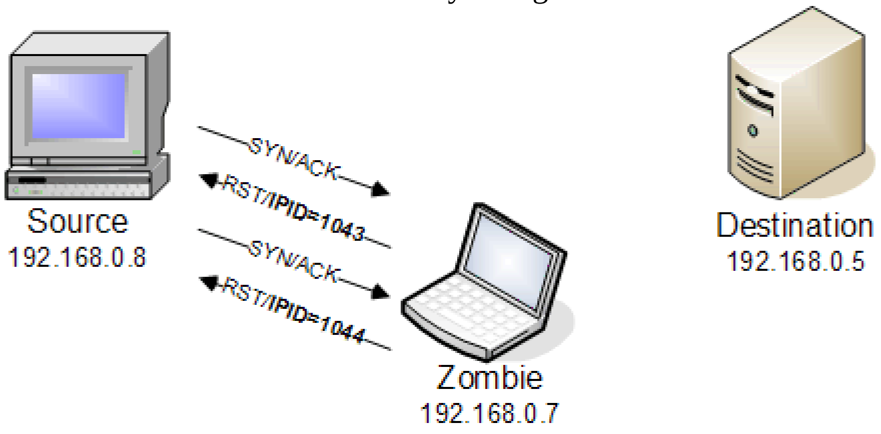
```
[192.168.0.7] [192.168.0.8] TCP: D=62803 S=80 RST WIN=0 / IP: IPID=1040
<--
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62962 SYN ACK=3093618703
SEQ=717494203 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.8] TCP: D=62962 S=80 RST WIN=0 / IP: IPID=1041
<--
```

Having clearly identified the group containing the mystery port, nmap begins the divide and conquer process again. This time, the list is split into a group of eight ports and another group containing seven ports. The group of eight ports is the first group to check:



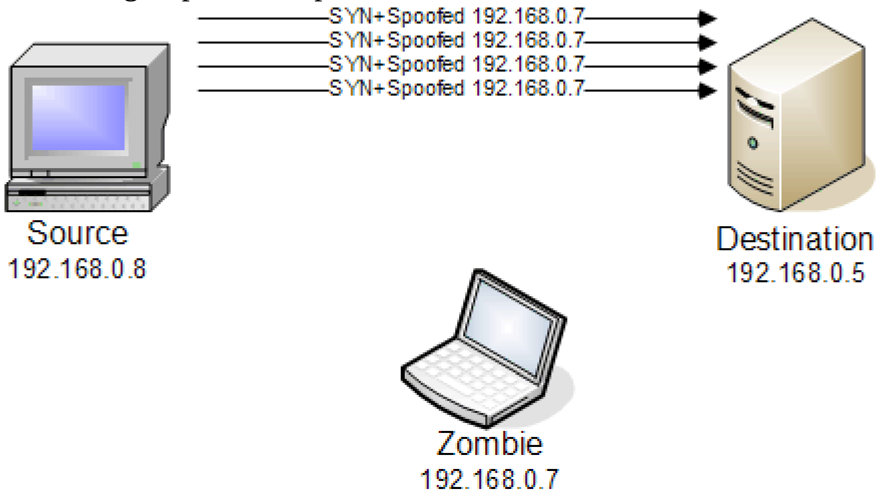
```
Source          Destination    Summary
----------------------------------------------------------------------
[192.168.0.7] [192.168.0.5] TCP: D=546 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=624 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=3072
[192.168.0.7] [192.168.0.5] TCP: D=188 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=6146 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=7007 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=1436 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=1234 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.5] [192.168.0.7] TCP: D=80 S=546 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=624 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=188 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=6146 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=7007 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=135 SYN ACK=3699814102
SEQ=2088591538 LEN=0 WIN=65535
[192.168.0.5] [192.168.0.7] TCP: D=80 S=1436 RST ACK=3699814102 WIN=0
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 RST WIN=0 / IP: IPID=1042
<--
[192.168.0.5] [192.168.0.7] TCP: D=80 S=1234 RST ACK=3699814102 WIN=0
```

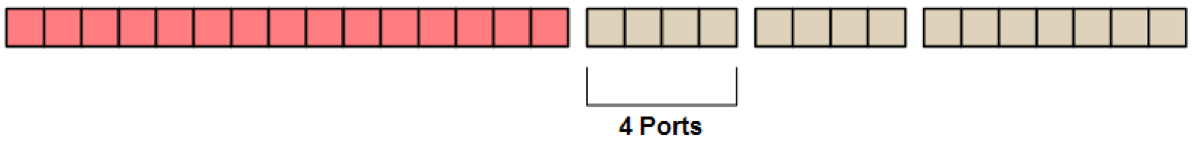
Nmap performs the normal IPID check for any changes:



```
Source         Destination   Summary
----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62867 SYN ACK=3093618703
SEQ=717494703 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.8] TCP: D=62867 S=80 RST WIN=0 / IP: IPID=1043
<--
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62735 SYN ACK=3093618703
SEQ=717495203 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.8] TCP: D=62735 S=80 RST WIN=0 / IP: IPID=1044
<--
```

The IPID incremented, so nmap knows the port is in the block of eight ports. Nmap now breaks this into two groups of four ports, and scans the first four:

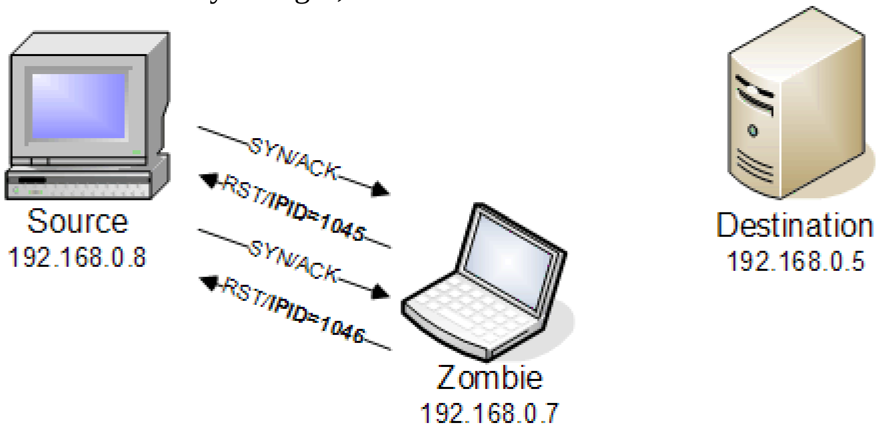




```
Source         Destination   Summary
-----------------------------------------------------------------------
[192.168.0.7] [192.168.0.5] TCP: D=546 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=624 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=188 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=6146 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.5] [192.168.0.7] TCP: D=80 S=546 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=624 RST ACK=3699814102 WIN=0
```

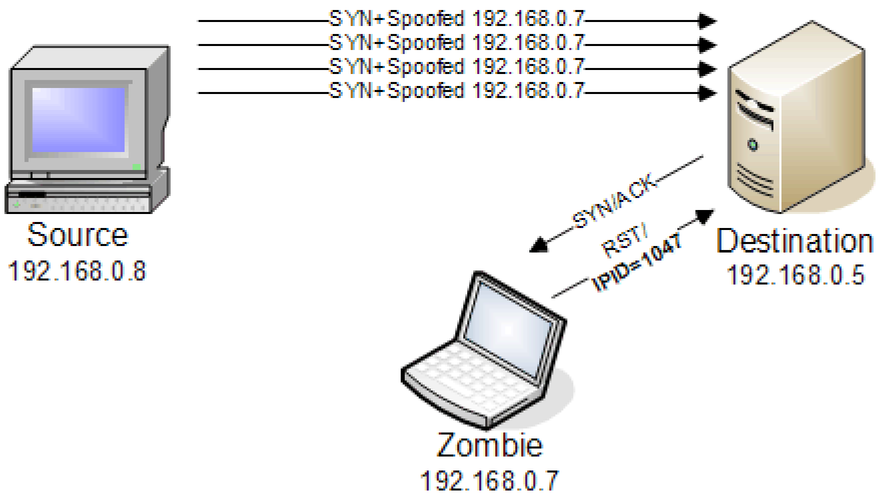
```
[192.168.0.5] [192.168.0.7] TCP: D=80 S=188 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=6146 RST ACK=3699814102 WIN=0
```

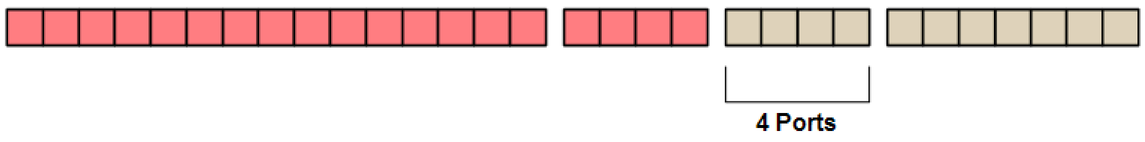
The IPID is checked for any changes, but it hasn't incremented since the last check:



```
Source          Destination     Summary
-----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62884 SYN ACK=3093618703
SEQ=717495703 LEN=0 WIN=3072
[192.168.0.7] [192.168.0.8] TCP: D=62884 S=80 RST WIN=0 / IP: IPID=1045
<--
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62919 SYN ACK=3093618703
SEQ=717496203 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.8] TCP: D=62919 S=80 RST WIN=0 / IP: IPID=1046
<--
```

After hundreds of frames, nmap has now narrowed down the open port to these last four possibilities. Instead of separating this group of four ports into a smaller query, nmap not-so-efficiently queries all four of the remaining ports:

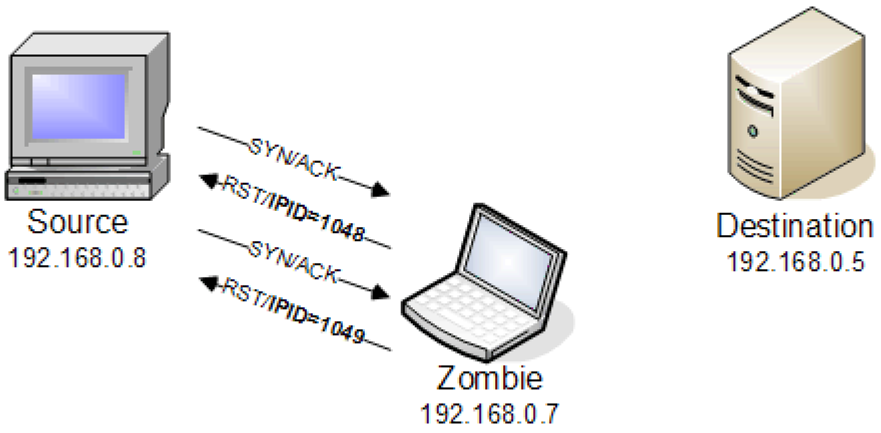




```
Source          Destination     Summary
-----------------------------------------------------------------------
[192.168.0.7] [192.168.0.5] TCP: D=7007 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=1436 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
```

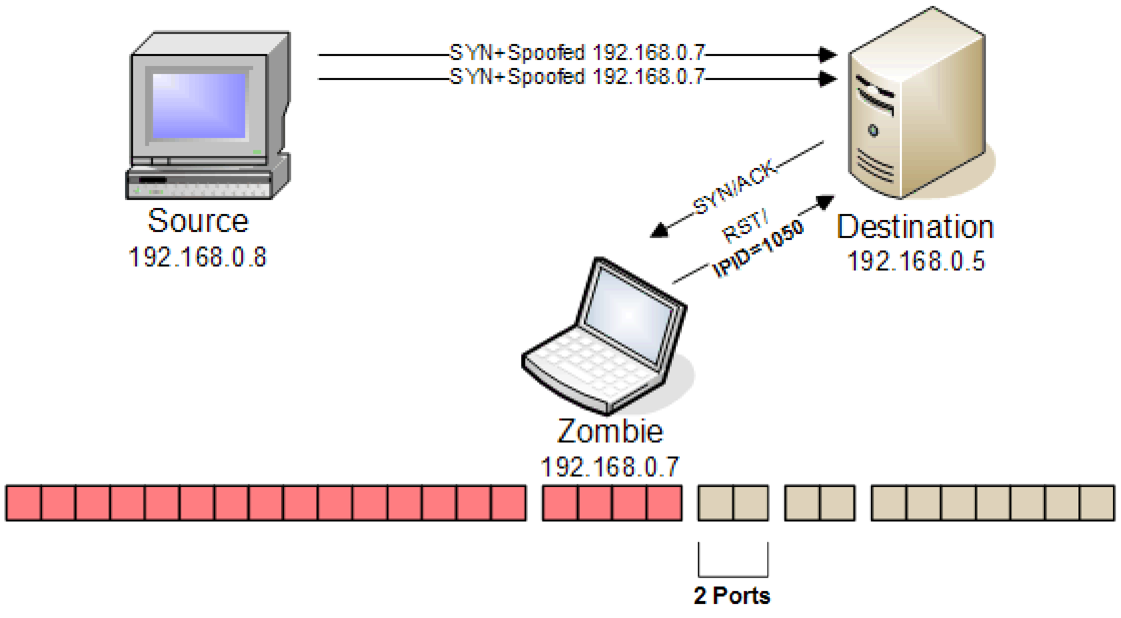
```
[192.168.0.7] [192.168.0.5] TCP: D=1234 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.5] [192.168.0.7] TCP: D=80 S=7007 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=135 SYN ACK=3699814102
SEQ=2088668207 LEN=0 WIN=65535
[192.168.0.5] [192.168.0.7] TCP: D=80 S=1436 RST ACK=3699814102 WIN=0
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 RST WIN=0 / IP: IPID=1047
<--
[192.168.0.5] [192.168.0.7] TCP: D=80 S=1234 RST ACK=3699814102 WIN=0
```

As expected, the IPID increments to confirm that the mystery port is contained in that last group of four ports:



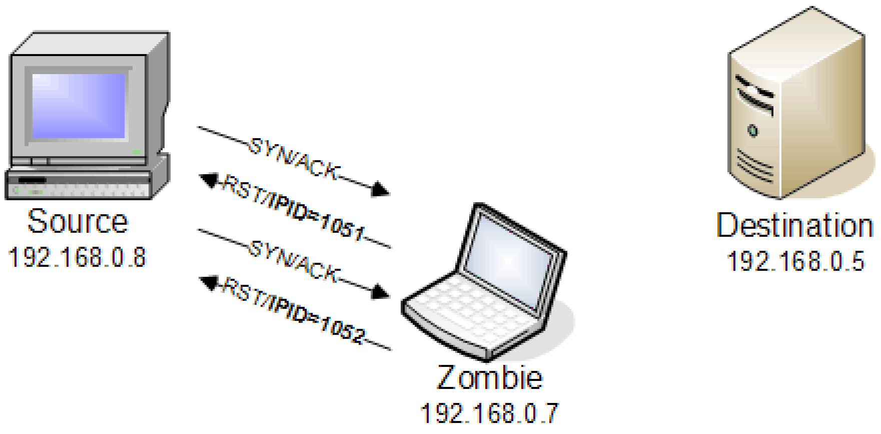
```
Source          Destination     Summary
----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62938 SYN ACK=3093618703
SEQ=717496703 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.8] TCP: D=62938 S=80 RST WIN=0 / IP: IPID=1048
<--
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62834 SYN ACK=3093618703
SEQ=717497203 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.8] TCP: D=62834 S=80 RST WIN=0 / IP: IPID=1049
<--
```

Nmap finally separates the four ports into two groups of two ports each and begins scanning the first two ports:



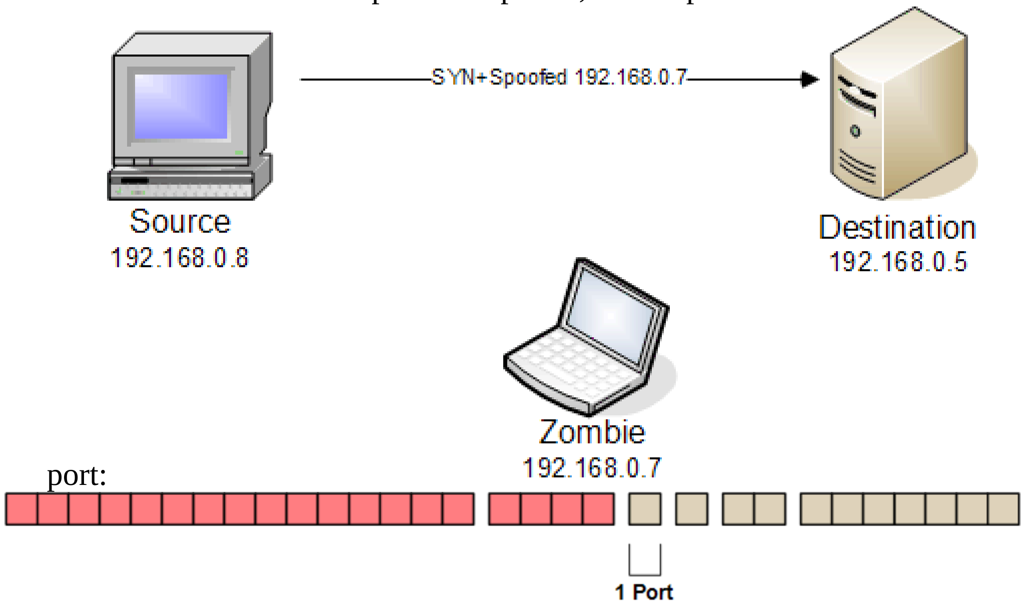
```
Source         Destination   Summary
-----------------------------------------------------------------------
[192.168.0.7] [192.168.0.5] TCP: D=7007 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.5] [192.168.0.7] TCP: D=80 S=7007 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=135 SYN ACK=3699814102
SEQ=2088695447 LEN=0 WIN=65535
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 RST WIN=0 / IP: IPID=1050
<--
```

The IPID check occurs again, and nmap discovers that the mystery open port is one of those two ports:

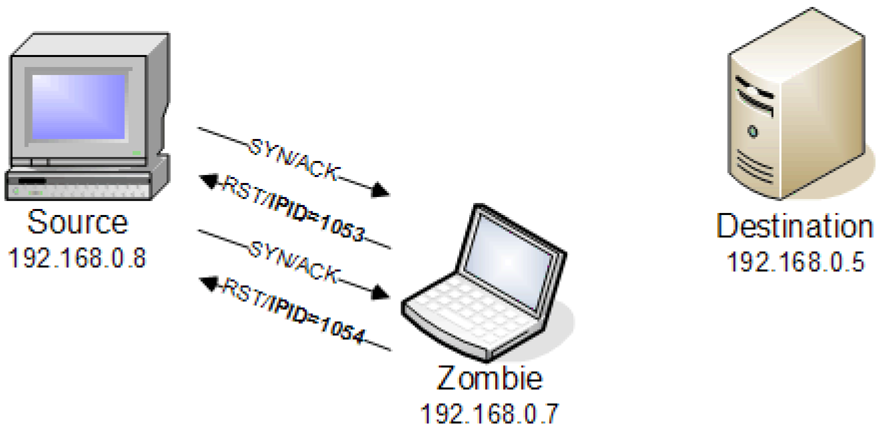


```
Source         Destination   Summary
-----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62729 SYN ACK=3093618703
SEQ=717497703 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.8] TCP: D=62729 S=80 RST WIN=0 / IP: IPID=1051
<--
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62937 SYN ACK=3093618703
SEQ=717498203 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.8] TCP: D=62937 S=80 RST WIN=0 / IP: IPID=1052
<--
```

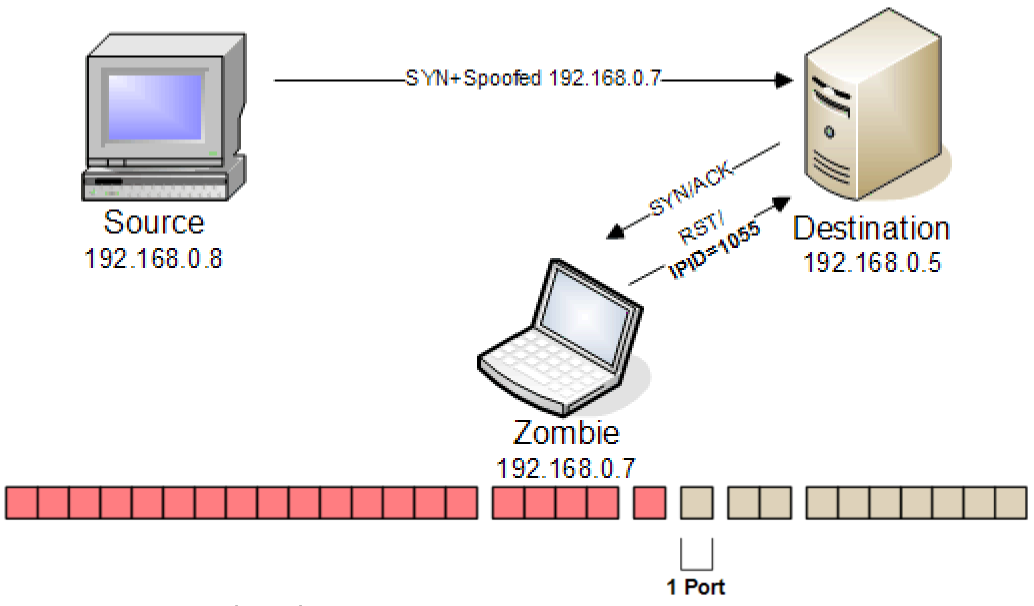
We're down to two possible options, so nmap checks the first port:



```
Source         Destination    Summary
----------------------------------------------------------------------
[192.168.0.7] [192.168.0.5] TCP: D=7007 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.5] [192.168.0.7] TCP: D=80 S=7007 RST ACK=3699814102 WIN=0
```
The resulting IPID check shows that the mystery port was not port 7007:



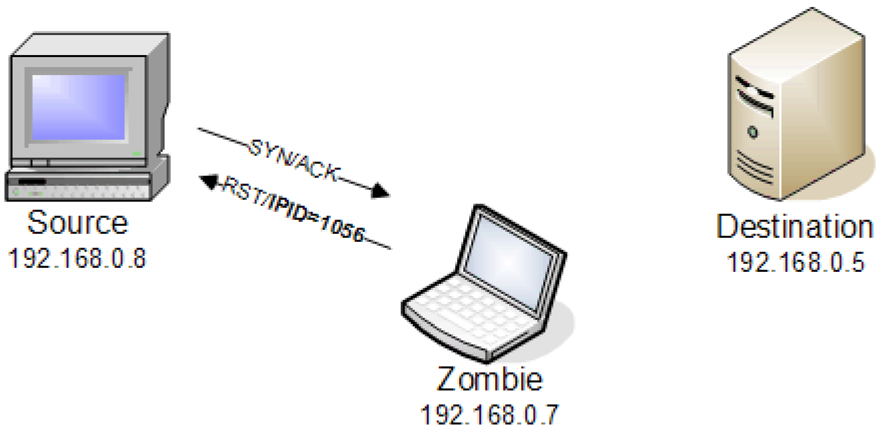
```
Source         Destination    Summary
----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62873 SYN ACK=3093618703
SEQ=717498703 LEN=0 WIN=3072
[192.168.0.7] [192.168.0.8] TCP: D=62873 S=80 RST WIN=0 / IP: IPID=1053
<--
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62737 SYN ACK=3093618703
SEQ=717499203 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.8] TCP: D=62737 S=80 RST WIN=0 / IP: IPID=1054
<--
```
At this point, there's one port remaining from the original thirty. Unfortunately, nmap's logic for idlescan doesn't recognize that there's only one choice remaining. So, nmap goes through the unnecessary motions of checking the final port:



```
Source         Destination    Summary
----------------------------------------------------------------------
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 SYN (Retransmission of
Frame 56) SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.5] [192.168.0.7] TCP: D=80 S=135 SYN ACK=3699814102
SEQ=2088767793 LEN=0 WIN=65535
```

```
[192.168.0.7] [192.168.0.5] TCP: D=135 S=80 RST WIN=0 / IP: IPID=1055
<--
```

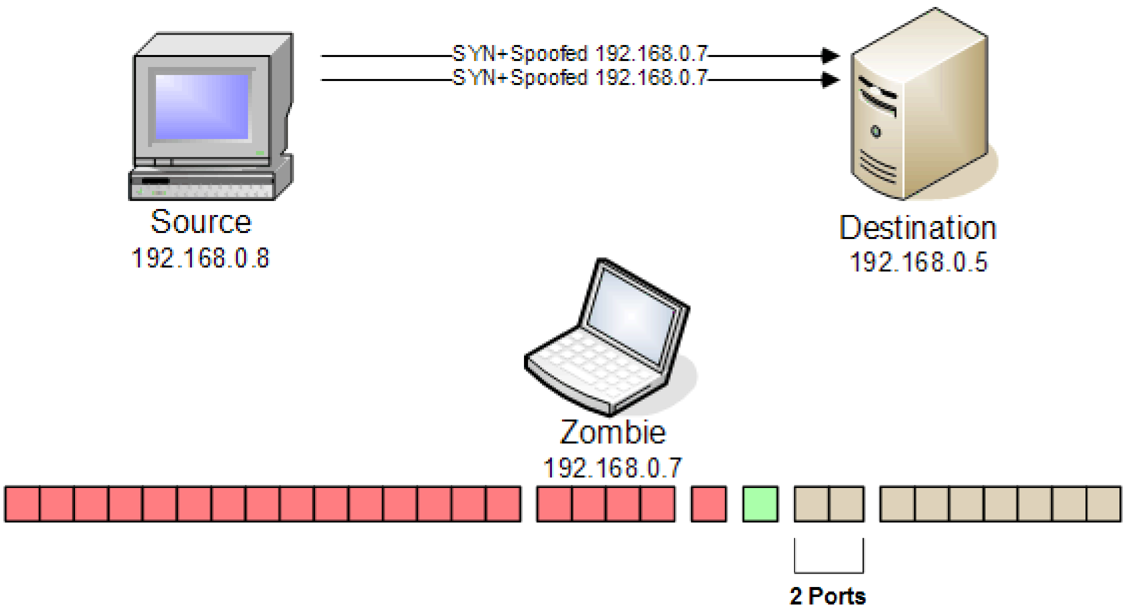
Unlike previous IPID checks, nmap only sends a single SYN/ACK for this IPID check. This is another inconsistency in the IPID process:



```
Source         Destination    Summary
-----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62782 SYN ACK=3093618703
SEQ=717499703 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.8] TCP: D=62782 S=80 RST WIN=0 / IP: IPID=1056
<--
```

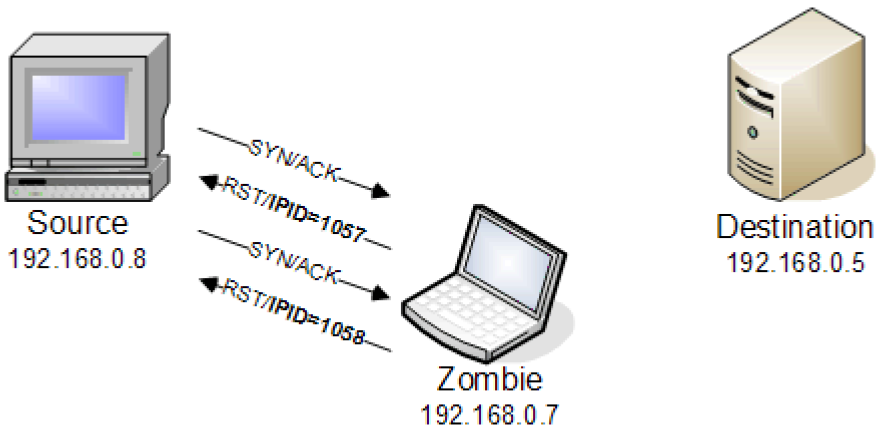
Finally, nmap has identified the single open port from the original group of thirty ports. Logically, nmap should continue to the next group of thirty ports and work through this same process. However, that's not what happens.

Remember when the group of four ports was split into two groups of two ports? Those other two ports were never scanned. Although it's now obvious that nmap has already found the open port, nmap still feels compelled to check those other two ports:



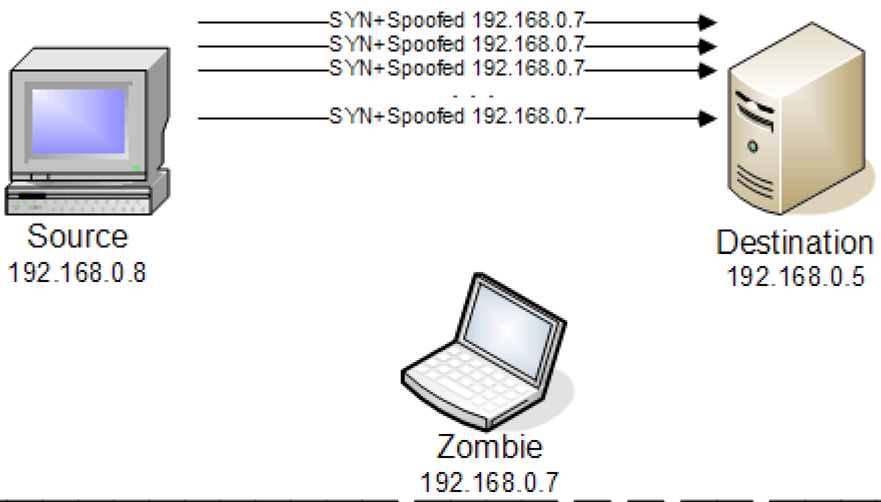
```
Source         Destination   Summary
-----------------------------------------------------------------------
[192.168.0.7] [192.168.0.5] TCP: D=1436 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=3072
[192.168.0.7] [192.168.0.5] TCP: D=1234 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.5] [192.168.0.7] TCP: D=80 S=1436 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=1234 RST ACK=3699814102 WIN=0
```

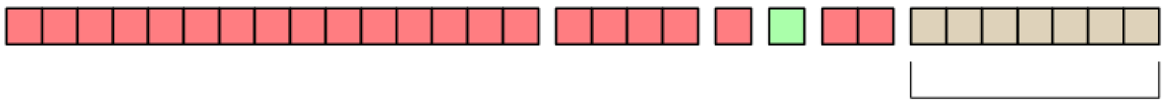
The IPID check shows that the open port wasn't in the last group, but we knew that already. Notice that nmap has returned to using two SYN/ACK packets to check the IPID:



```
Source         Destination   Summary
-----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62875 SYN ACK=3093618703
SEQ=717500203 LEN=0 WIN=3072
[192.168.0.7] [192.168.0.8] TCP: D=62875 S=80 RST WIN=0 / IP: IPID=1057
<--
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62729 SYN ACK=3093618703
SEQ=717500703 LEN=0 WIN=3072
[192.168.0.7] [192.168.0.8] TCP: D=62729 S=80 RST WIN=0 / IP: IPID=1058
<--
```

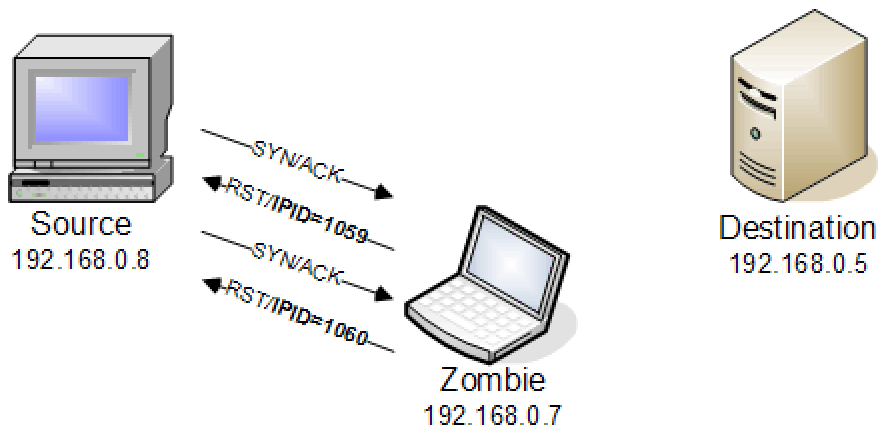
Although the open port was identified a few IPID checks ago, nmap remembers that there are still seven ports that haven't been scanned. Although it's not necessary at this point, nmap insists on scanning these ports:



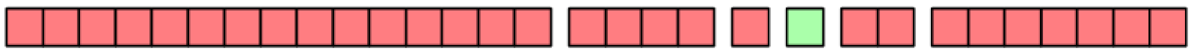


```
Source         Destination    Summary
----------------------------------------------------------------------
[192.168.0.7] [192.168.0.5] TCP: D=808 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=665 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=790 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.5] TCP: D=873 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=3072
[192.168.0.7] [192.168.0.5] TCP: D=717 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.5] TCP: D=185 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=1024
[192.168.0.7] [192.168.0.5] TCP: D=569 S=80 SYN (Retransmission)
SEQ=3699814101 LEN=0 WIN=3072
[192.168.0.5] [192.168.0.7] TCP: D=80 S=808 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=665 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=790 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=873 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=717 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=185 RST ACK=3699814102 WIN=0
[192.168.0.5] [192.168.0.7] TCP: D=80 S=569 RST ACK=3699814102 WIN=0
```

Obviously, this IPID check shows that these last seven ports weren't open:



```
Source         Destination    Summary
----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62726 SYN ACK=3093618703
SEQ=717501203 LEN=0 WIN=4096
[192.168.0.7] [192.168.0.8] TCP: D=62726 S=80 RST WIN=0 / IP: IPID=1059
<--
[192.168.0.8] [192.168.0.7] TCP: D=80 S=62763 SYN ACK=3093618703
SEQ=717501703 LEN=0 WIN=3072
[192.168.0.7] [192.168.0.8] TCP: D=62763 S=80 RST WIN=0 / IP: IPID=1060
<--
```



Finally, it's over. The first thirty ports have been scanned, and now nmap groups together another thirty random ports and repeats the process again until all of the ports have been checked.

Advantages of Idlescan
Idlescan's method of spoofing IP addresses and checking IPIDs allows nmap to find open ports from a distance, even if packet filters are in place! Nmap simply requires any open port to a zombie workstation to complete the communication process.

One of the largest advantages of idlescan is the stealth factor. A destination station will never see the IP address of the nmap station!

Disadvantages of Idlescan
Idlescan only locates ports. Idlescan can't provide any application version information or operating system fingerprinting.

The zombie must be an idle station, and that's difficult to know for sure. Often, many different devices will be tested with idlescan before an appropriate idle station is identified.

The divide-and-conquer method of idlescan's bulk processing means that there will be more network traffic than a normal port scan. In most cases, many ports will be scanned more than once. The idlescan logic also scans more ports than necessary, inefficiently using network bandwidth and extending the scanning time.

Although Idlescan spoofs the IP address of the zombie, this method of invisibility isn't helpful if all of the devices are on the same IP subnet. Since the MAC address of the nmap station isn't modified, a sharp eye can pick the spoofed address out of a network trace. To maintain the stealth-factor, the nmap station and the destination should be on separate IP subnets.

Idlescan requires privileged access to create the spoofed IP frames. Without privileged access, this scan will not run.

When to use Idlescan Idlescan allows the nmap station to remain hidden from the remote station. If invisibility is important, idlescan can provide a bit of cover while poking around the network.

Since idlescan only needs a single port to a zombie workstation, it's relatively simple to scan inside of a protected network using the zombie as a scanning proxy. This also allows nmap to check trust relationships inside an internal network without requiring direct communication between the devices.

Fyodor has written a fantastic idlescan and IPID white paper, located at:

http://www.insecure.org/nmap/idlescan.html

Idlescan has the potential to extract a lot of information about your network, in spite of your packet-filters! If you are concerned about idlescan extracting information through your firewall, use the operating system fingerprinting option (`-O`) to test the IPID predictability of your systems.

FTP Bounce Attack(`-b <ftp_relay_host>`)

Requires Privileged Access: NO
Identifies TCP Ports: YES
Identifies UDP Ports: NO

The FTP bounce attack is infamous in the network security world. The modern threat associated with this attack methodology has been nullified by the retooling of most FTP services, rendering this particular attack more interesting for its technical process than for its potential maliciousness. Like idlescan, the FTP bounce attack uses a third workstation to act as a proxy between the nmap host and the destination station.

A popular detailed discussion of the ftp bounce attack was written by "Hobbit" on the Bugtraq mailing list. A copy of the message can be found at this location:

http://www.insecure.org/nmap/hobbit.ftpbounce.txt

The FTP bounce attack uses an FTP server in passive mode to transmit information to any device on the network. For a detailed explanation of active mode FTP vs. passive mode FTP, refer to this site:

http://slacksite.com/other/ftp.html

FTP Bounce Attack Operation

The FTP bounce attack wouldn't be possible if it weren't for passive mode FTP. With passive mode FTP, the command connections are completely separate from the data connections. This allows the FTP server to work well with firewalls because the FTP server is responsible for building the outbound data connection with the remote host. However, it also means that a user could send a PORT command to an FTP server that would direct the data towards a completely different host!

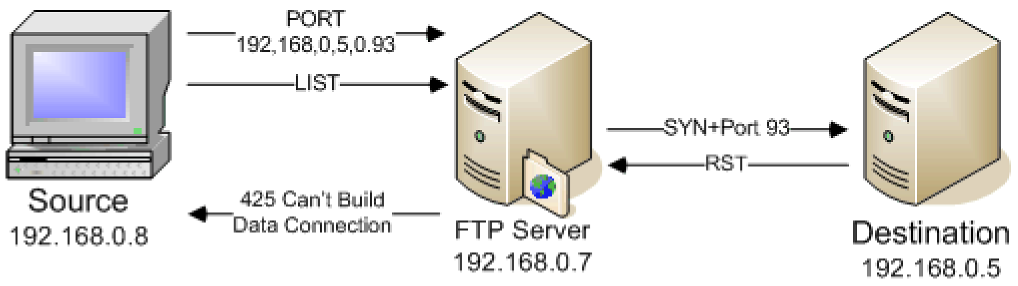
From a security perspective, a "bounceable" FTP server is a serious concern. For the purposes of port scanning, however, this situation couldn't be more convenient! The FTP bounce attack takes advantage of these poorly-configured FTP servers to provide nmap with a unique method of locating open ports.

To begin the bounce attack process, nmap must login to the FTP server that will be used as the "middleman." Once connected to the FTP server, nmap sends the PORT command to direct all data connections to the destination IP address and TCP port.

The PORT command has a unique syntax. The PORT command is followed by six numbers that are separated by commas. The first four numbers refer to the four octets of the destination IP address, and the last two numbers refer to the port number on the remote device. To calculate the port number into decimal, multiply the second-to-last number by 256 and add it to the last number. For example, the command `PORT 192,168,0,5,2,44` refers to IP address 192.168.0.5 and port (2*256)+44, or port 556.

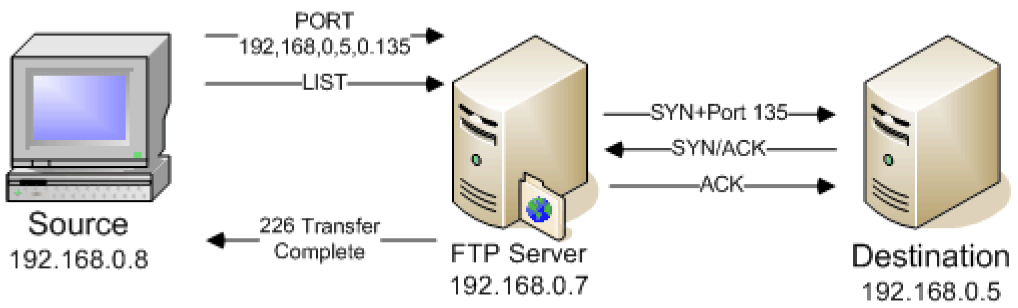
Once the nmap source specifies the port command, it sends a LIST command to launch the data connection over the specified IP address and TCP port. The FTP server then attempts a connection with the device specified in the PORT command.

A closed port will result with the FTP server informing the source station that the FTP server can't build the connection:



```
Source         Destination    Summary
----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] FTP: C PORT=37205   PORT 192,168,0,5,0,93
[192.168.0.7] [192.168.0.8] FTP: R PORT=37205   200 PORT command
successful.
[192.168.0.8] [192.168.0.7] FTP: C PORT=37205   LIST
[192.168.0.7] [192.168.0.5] TCP: D=93 S=20 SYN SEQ=474501024 LEN=0
WIN=65535
[192.168.0.5] [192.168.0.7] TCP: D=20 S=93 RST ACK=474501025 WIN=0
[192.168.0.7] [192.168.0.8] FTP: R PORT=37205   425 Can't build data
connection: Connection refused.
```

An open port completes the transfer over the specified connection:



```
Source         Destination    Summary
-----------------------------------------------------------------------
[192.168.0.8] [192.168.0.7] FTP: C PORT=37205   PORT 192,168,0,5,0,135
[192.168.0.7] [192.168.0.8] FTP: R PORT=37205   200 PORT command
successful.
[192.168.0.8] [192.168.0.7] FTP: C PORT=37205   LIST
[192.168.0.7] [192.168.0.5] TCP: D=135 S=20 SYN SEQ=4240951199 LEN=0
WIN=65535
[192.168.0.5] [192.168.0.7] TCP: D=20 S=135 SYN ACK=4240951200
SEQ=2193395373 LEN=0 WIN=65535
[192.168.0.7] [192.168.0.5] TCP: D=135 S=20     ACK=2193395374
WIN<<1=65700
[192.168.0.7] [192.168.0.8] FTP: R PORT=37205   150 Opening ASCII mode
data connection for '/bin/ls'.
[192.168.0.7] [192.168.0.8] FTP: R PORT=37205   226 Transfer complete.
[192.168.0.7] [192.168.0.5] FTP: R PORT=135   Text Data
```

The nmap output shows the results of the FTP bounce scan. Since the bounce scan is often performed through firewalls, nmap adds a reminder to include the "don't ping" option (`-P0`) on the command line.

```
# nmap -v -b anonymous:anon@192.168.0.7 192.168.0.5
```

```
Hint: if your bounce scan target hosts aren't reachable from here,
remember to
use -P0 so we don't try and ping them prior to the scan

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-04-23
20:37 EDT
Resolved ftp bounce attack proxy to 192.168.0.7 (192.168.0.7).
Attempting connection to ftp://anonymous:anon@192.168.0.7:21
Connected:Login credentials accepted by ftp server!
Initiating TCP ftp bounce scan against 192.168.0.5 at 20:37
Discovered open port 6969/tcp on 192.168.0.5
Discovered open port 135/tcp on 192.168.0.5
Discovered open port 139/tcp on 192.168.0.5
Discovered open port 445/tcp on 192.168.0.5
Scanned 1663 ports in 9 seconds via the Bounce scan.
Host 192.168.0.5 appears to be up ... good.
Interesting ports on 192.168.0.5:
(The 1659 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
6969/tcp open  acmsoda
MAC Address: 00:11:43:43:A8:34 (Dell   (WW Pcba Test))

Nmap finished: 1 IP address (1 host up) scanned in 20.602 seconds
               Raw packets sent: 2 (68B) | Rcvd: 1 (46B)
#
```

Advantages of the FTP Bounce Attack
Like idlescan, the FTP bounce attack can scan "through" a firewall. The nmap station needs only an FTP login to scan any device that can be accessed from the FTP server. This is a significant feature, since reconnaissance of the protected network would not be possible otherwise.
The FTP bounce attack uses standard FTP functionality. Nmap does not require specialized packet configurations or changes to the FTP protocol. Therefore, the FTP bounce attack does not require any privileged access.

Disadvantages of the FTP Bounce Attack
The largest disadvantage of the FTP bounce attack relates to the availability of an FTP server that allows a PORT command to redirect the data connection to a third device. Most FTP servers have their default configuration to protect against this use of the PORT command, although it technically that modification does not follow the FTP standard.

My FreeBSD server had to be reconfigured through `inetd.conf` to start `ftpd` with the `–R` parameter to disable the protection and strictly comply with the FTP RFC. This is obviously not the normal configuration for a production FTP server!

The FTP bounce attack can only scan TCP ports. Since FTP doesn't connect to remote devices with UDP, it's not possible to get feedback on the availability of UDP ports.

The process of bouncing through an FTP server is slow when compared to other scanning methods. The port scanning requests can only check a single port at a time, and the current nmap bounce attack options only provide for a single FTP connection.

The FTP bounce attack starts an application session with the FTP server, and most FTP servers will log the connection and all of the commands used during the session.

When to use the FTP Bounce Attack The FTP bounce attack is a well positioned TCP port scan through a firewall. FTP is a commonly available application through a packet-filtering device, and a connection to an FTP server provides the perfect jumping-off-point to gather more information about the rest of the protected network. If the FTP server is this poorly maintained, there are probably more devices that need to be identified and corrected!

The FTP bounce attack does not provide version information, but there is potential to create additional functionality. Adding this functionality in nmap is questionable, however, because there are few FTP devices available where a bounce attack would be applicable (and the numbers are dwindling). There are many more potential enhancements to nmap that would provide much more capability than updating the FTP bounce attack.

The FTP bounce attack is interesting, but it's probably not going to work with contemporary FTP servers. If you need to scan through a firewall, you may have better luck with idlescan.

Chapter 3:

Nmap's Ping Options

Understanding nmap's myriad scan types is only the beginning of harnessing its power. Nmap's additional options provide over fifty different choices of packet timings, ping options, output formats, and other customizable features! Although this considerable quantity of options seems overwhelming, it's this abundance of choices that provides nmap with incredible flexibility in nearly any networking environment.

Ping Options

Nmap always "pings" a remote station before initiating the scanning process. The default nmap ping consists of an ICMP echo request followed by a TCP ACK on port 80. If a station does not respond to either ping method, nmap will continue to the next target. If the scan does not have any additional targets, the scan will end.

Networking purists consider the term "ping" as a reference to an ICMP echo request and the corresponding ICMP echo reply. However, nmap's use of the word "ping" is more generic. In the nmap world, a ping is any request that would prompt a remote station's response. Throughout this text, a ping will refer to nmap's more relaxed definition.

The purpose of an nmap ping is to provoke any kind of response from a remote station.

Once a response is received from a remote device, nmap identifies that device as active on the network and begins scanning it for detailed port information. Most of these ping options can be combined together to maximize the possibility of locating a device through firewalls or packet filters.

Nmap's pings can also be customized for the situation. For example, a firewall that blocks ICMP and ACK on port 80 might allow nmap to ping through the firewall with a TCP SYN to port 135 or a UDP query to port 22. This customization could also be used as a filter that would only identify devices that fit certain profiles, such as routers or mail servers.

Subnet Broadcasts
If an IP subnet is selected as a destination, nmap will also send ping requests to the subnet's broadcast addresses. For example, an nmap destination of 192.168.0.2/24 will prompt an nmap ping of all hosts between 192.168.0.1 and 192.168.0.254. Additionally, nmap will include 192.168.0.0 and 192.168.0.255 as destination addresses.

Many systems will respond to an "all-zeros" or "all-ones" broadcast, even though an individual IP address has not been specified. Some operating systems have been configured to ignore these types of broadcasts, but many stations will unknowingly respond to at least one of these broadcast types.

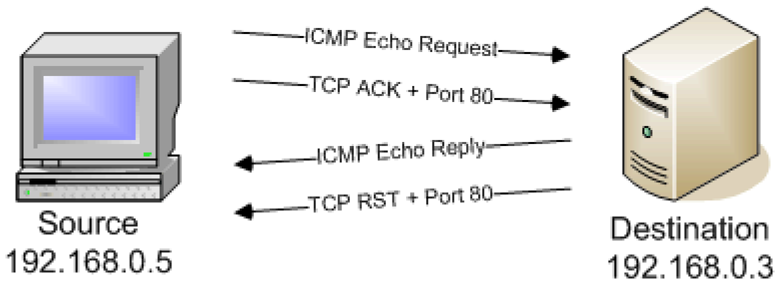
ICMP Echo Request and TCP ACK Ping (`-PB`)

THIS OPTION HAS BEEN DEPRECATED
If a specific ping type is not specified on the command line, the default nmap ping is used. This default ping consists of an ICMP echo request followed by a TCP ACK.

The TCP ACK is always sent, even if an ICMP echo response is received from the remote device before the TCP ACK is sent.

The `–PB` option refers to the default combination of an ICMP echo request and a TCP ACK. This option is being phased out in favor of a separate ICMP echo request ping option (`-PE`) and a separate TCP ACK ping option (`-PA`). Now that these ping methods have separate options, they can be used independently of each other or in combination with other ping options. With this added flexibility, this `–PB` option that combines the ping methods is redundant and unnecessary for future use.

ICMP Echo Request and TCP ACK Ping Operation
The ICMP echo request and TCP ACK requests are both sent together and nmap waits to receive a reply from one or both of the requests:



```
Source         Destination    Summary
-------------------------------------------------------------------------
[192.168.0.5] [192.168.0.3] ICMP: Echo
[192.168.0.5] [192.168.0.3] TCP: D=80 S=43162     ACK=1866185566
WIN=1024
[192.168.0.3] [192.168.0.5] ICMP: Echo reply
[192.168.0.3] [192.168.0.5] TCP: D=43162 S=80 RST WIN=0
```

Advantages of the ICMP Echo Request and TCP ACK Ping

The ICMP echo request and TCP ACK ping is a good all-purpose ping. This ping will obtain a response from all devices if the path is clear for ICMP, and it will get a response from a remote device if port 80 is not filtered. Since most packet filters allow port 80, there's a good possibility that this packet will get through.

It doesn't matter that a remote station isn't running a web server on port 80, since all stations receiving a 'surprise' ACK packet will reply with a RST frame!

Disadvantages of the ICMP Echo Request and TCP ACK Ping

ICMP is often filtered through firewalls or packet filters. This random ACK ping through a stateful firewall won't usually garner a response. In some cases, a scan against a series of hosts may not return any available systems because the default ping couldn't traverse the firewall.

When to use the ICMP Echo Request and TCP ACK Ping

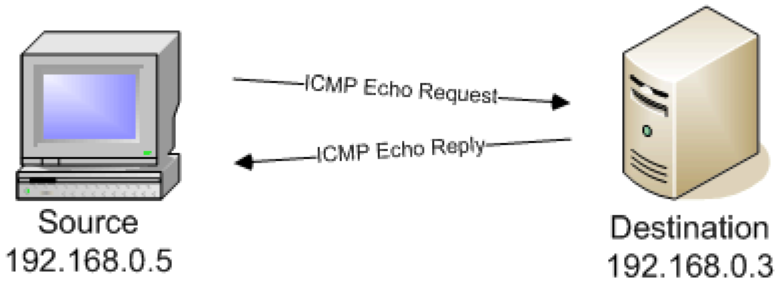
DON'T USE THIS OPTION. The `-PB` option has been deprecated, which means that this option will be eventually phased out in a future release of nmap. It still works in nmap 3.81, but its future availability isn't guaranteed. As a replacement, use both the ICMP echo request ping (`-PE`) and TCP ACK ping (`-PA`) options together on the nmap command line.

ICMP Echo Request Ping (-PE)

The ICMP echo request ping sends an ICMP echo request to nmap's destination IP address. This is the same type of ICMP echo request that is sent in the default ICMP echo request and TCP ACK ping combination. With the `-PE` option, the ICMP echo request can be specified as the nmap ping method without pairing it with the TCP ACK ping.

The -PI option is an undocumented alias for the ICMP echo request ping.

ICMP Echo Request Ping Operation
This ping method consists of a single ICMP echo request frame. If an ICMP echo reply frame is received in return, the station is active:



```
Source         Destination    Summary
------------------------------------------------------------------------
[192.168.0.5] [192.168.0.3] ICMP: Echo
[192.168.0.3] [192.168.0.5] ICMP: Echo reply
```

If a response is not received to the ICMP echo request, the IP address is not active or the connection to the remote device is filtered.

Advantages of the ICMP Echo Request Ping
An ICMP echo request is a simple and very common method of determining a station's availability. Many network administrators use this method to determine station availability through the aptly named "ping" command.

If ICMP traffic isn't filtered between the nmap station and the destination device, the entire network connection between the two stations may have very little filtering. It's very common for ICMP to be filtered through firewalls or packet filters, so a response to an ICMP echo request would be indicative of a link that has very little (if any) filtering.

The ICMP echo doesn't rely on any particular application or open port to work. If a remote device communicates via TCP/IP, then it's most often a candidate for the ICMP echo request ping.

Disadvantages of the ICMP Echo Request Ping
As mentioned above, ICMP is one of the most filtered protocols in enterprise networks. The ICMP protocol has the ability to redirect traffic, identify available workstations, and identify closed ports on a device. When a firewall or packet filter is installed, one of the first rules is the restriction of ICMP.

When to use the ICMP Echo Request Ping
If an nmap scan runs within a local network, then ICMP will probably be a successful ping method. As long as the network link is unfiltered, an ICMP echo request is a very reliable determination of station availability. If there's any doubt regarding the free flow of packets between the nmap scanning station and the remote device, a different or additional nmap ping type should be specified.

TCP ACK Ping (`-PA [portlist]`, `-PT [portlist]`)

Instead of using the default option of both an ICMP echo request and a TCP ACK, the `–PA` option sends a TCP ACK and forgoes any ICMP echo requests. This is a good alternative when the use of ICMP is not applicable because of packet filtering or firewalls.

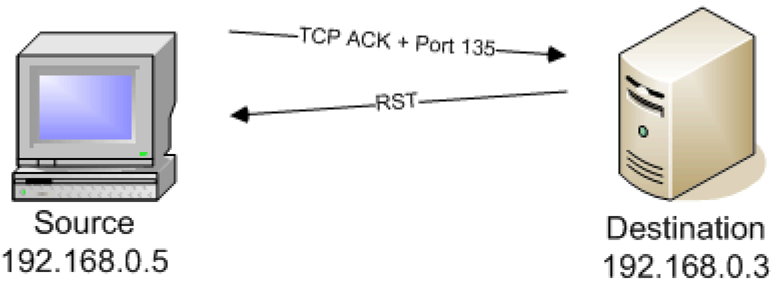
TCP ACK Ping Operation
The TCP ACK ping consists of a random TCP ACK sent to a remote device. If the device is active, a RST will be received in return. If the device is not active or the port is filtered, there will be no response to the ACK. It's interesting to note that these results are identical to the results found with the TCP ACK scan (`-sA`).

The `[portlist]` option allows the user to specify a series of ports for the ACK ping to use. The specification of this port list isn't as flexible as specifying IP addresses, but this is a pre-scan ping process and not an actual scan. For the ACK ping, the ports are listed individually with each port separated with a comma. For example, the command
```
# nmap –v –sS 10.0.0.4 –PA23,110
```
will attempt a TCP ACK ping to host 10.0.0.4 over ports 23 and 110. If neither of these ports replies to the ACK ping, the SYN scan will not run and the scan attempt will stop with this message:

```
Note: Host seems down. If it is really up, but blocking our
ping probes, try -P0
```

If a port is not specified, port 80 is used by default.

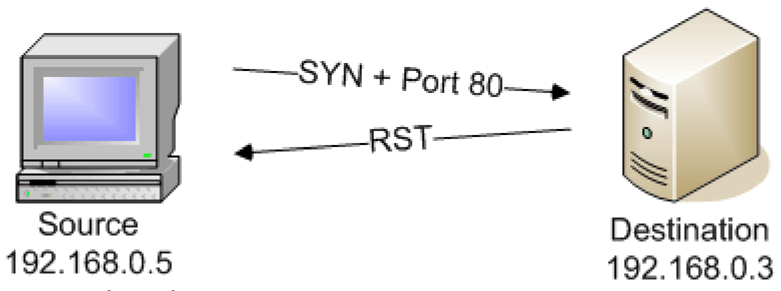
This is an example of a TCP ACK scan as a privileged user. In every case where a system is identified, a RST is returned. If no response is received, the destination IP address either isn't on the network or the station is behind a firewall:



```
Source          Destination   Summary
-----------------------------------------------------------------------
```

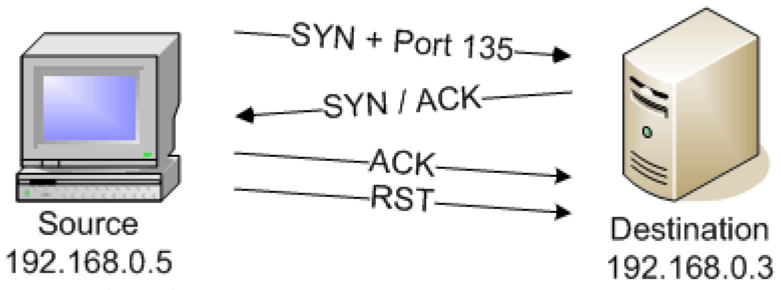
```
[192.168.0.5] [192.168.0.3] TCP: D=135 S=62984     ACK=3382473630
WIN=3072
[192.168.0.3] [192.168.0.5] TCP: D=62984 S=135 RST WIN=0
```

If the nmap user is not a privileged user, a TCP connect()-style ping is used instead of a TCP ACK. **This adjustment occurs without an error message or other notification!** This means that instead of a TCP ACK, the nmap station will send a TCP SYN. If the TCP SYN is sent to a closed port, a TCP RST is received in return:



```
Source         Destination    Summary
-----------------------------------------------------------------------
[192.168.0.5] [192.168.0.3] TCP: D=80 S=60554 SYN SEQ=2685844190 LEN=0
WIN=5840
[192.168.0.3] [192.168.0.5] TCP: D=60554 S=80 RST ACK=2685844191 WIN=0
```

If the TCP connect()-style ping is "lucky" enough to ping an open port, an application session is completely opened:



```
Source         Destination    Summary
----------------------------------------------------------------------
[192.168.0.5] [192.168.0.3] TCP: D=135 S=36960 SYN SEQ=219557237 LEN=0
WIN=5840
[192.168.0.3] [192.168.0.5] TCP: D=36960 S=135 SYN ACK=219557238
SEQ=4137096645 LEN=0 WIN=65535
[192.168.0.5] [192.168.0.3] TCP: D=135 S=36960     ACK=4137096646
WIN<<2=5840
[192.168.0.5] [192.168.0.3] TCP: D=135 S=36960 RST ACK=4137096646
WIN<<2=5840
```

To ensure that no application sessions could ever open, the TCP ACK ping must run as a privileged user!

Advantages of the TCP ACK Ping

The TCP ACK ping uses little network traffic. In most cases, the common ACK query to a device is relatively undetectable. The TCP ACK ping also allows different port numbers to be probed, providing nmap with additional options when negotiating a scan through a firewall.

The TCP ACK ping identifies a filtered connection or a non-filtered connection. This is a perfect use of a ping probe, since the goal of the ping is to get any type of response from a remote device. An nmap ping doesn't focus on identifying ports. Instead, its goal is to locate other active devices.

Disadvantages of the TCP ACK Ping

The TCP ACK ping is only available to privileged users. If a non-privileged user requests a TCP ACK ping, nmap 3.81 changes the request to a TCP connect() ping but doesn't provide any feedback or warning! These circumstances would be unfortunate for an nmap user that was interested in keeping a low profile by remaining out of a server's application logs. If the only option is to run as a non-privileged user, the nmap option should include a port number that would get through a firewall but would not be an active application port on a device.

When to use the TCP ACK Ping

Because of its low profile and flexible configuration options, the TCP ACK ping can be used in almost any circumstance. The ACK ping's port values can be modified, and multiple ports can be included on the command line.

It's important to note that the TCP ACK ping will only run "silently" if the nmap user is privileged. A non-privileged user's ping will perform a TCP connect() and initiate an application session if the requested port number matches an open port on the remote device.

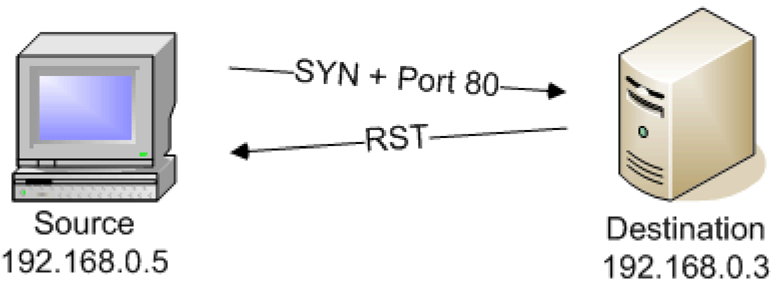
TCP SYN Ping (`-PS [portlist]`)

A TCP SYN ping uses the same process as nmap's TCP SYN scan to identify a remote device. With the TCP SYN ping, the nmap scanner is looking for an RST from a closed port or an ACK from an open port. Either result will provide nmap with proof that an active system resides at that destination IP address.

TCP SYN Ping Operation

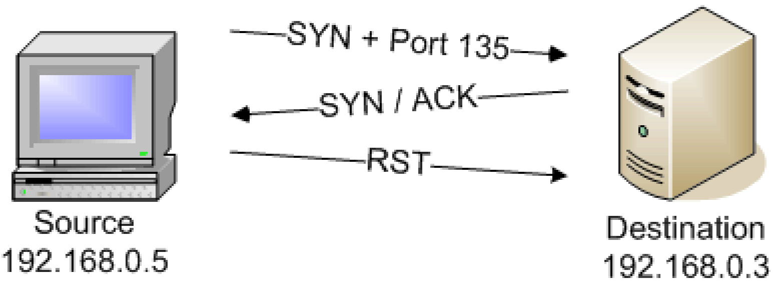
The response to a TCP SYN packet on a closed port is usually a RST, and if nmap is lucky enough to ping an open port it will receive a SYN/ACK (to which nmap immediately sends a RST). Of course, either response is all that nmap needs to recognize a system is alive on the other end!

The TCP SYN ping from a privileged user will receive a RST if the port is closed:



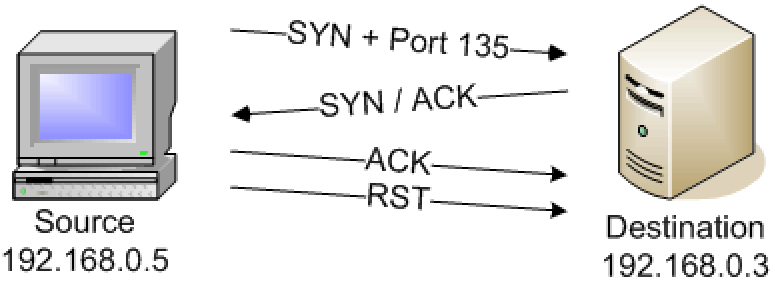
```
Source          Destination    Summary
-----------------------------------------------------------------------
---------------
[192.168.0.5] [192.168.0.3] TCP: D=80 S=44545 SYN SEQ=3017830046 LEN=0
WIN=4096
[192.168.0.3] [192.168.0.5] TCP: D=44545 S=80 RST ACK=3017830047 WIN=0
```

A TCP SYN ping from a privileged user will receive an ACK if the port is open, to which nmap will respond with a RST:



```
Source          Destination    Summary
-----------------------------------------------------------------------
---------------
[192.168.0.5] [192.168.0.3] TCP: D=135 S=40804 SYN SEQ=3181151454 LEN=0
WIN=2048
[192.168.0.3] [192.168.0.5] TCP: D=40804 S=135 SYN ACK=3181151455
SEQ=3723360488 LEN=0 WIN=65535
[192.168.0.5] [192.168.0.3] TCP: D=135 S=40804 RST WIN=0
```

If the TCP SYN ping type is requested by a non-privileged user, nmap will default to a TCP connect()-style ping. **This adjustment occurs without an error message or other notification!** This shows the results of a TCP connect()-style ping to an open port:



```
Source          Destination    Summary
-----------------------------------------------------------------------
[192.168.0.5] [192.168.0.3] TCP: D=135 S=46558 SYN SEQ=735305937 LEN=0
WIN=5840
[192.168.0.3] [192.168.0.5] TCP: D=46558 S=135 SYN ACK=735305938
SEQ=1935433798 LEN=0 WIN=65535
[192.168.0.5] [192.168.0.3] TCP: D=135 S=46558     ACK=1935433799
WIN<<2=5840
[192.168.0.5] [192.168.0.3] TCP: D=135 S=46558 RST ACK=1935433799
WIN<<2=5840
```

To prevent any possible initialization an application session, the TCP SYN ping option should always run as a privileged user.

Like the TCP ACK ping, the SYN ping can specify individual port numbers delimited with commas. If a port is not specified, port 80 is used by default.

Advantages of the TCP SYN Ping
The TCP SYN ping accomplishes its goal in just a few packets. This minimal amount of network traffic appears to be normal TCP handshake frames. This makes the TCP SYN ping appear almost invisible when compared to normal network traffic.

Disadvantages of the TCP SYN Ping
The TCP SYN ping should run as a privileged user to ensure that no application sessions are started. Unfortunately, the default nmap function for a non-privileged user is to use a TCP connect()-style ping. If the destination station has an open port that matches the TCP SYN ping port, an application session will be initialized and the session will be identified in the application logs.

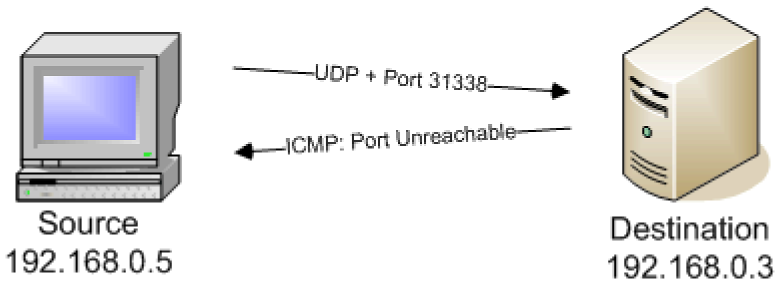
When to use the TCP SYN Ping
Although the TCP SYN ping can discern between an open port and a non-open port, the disposition doesn't matter as long as the device provides some kind of response. The TCP SYN ping is such a normal frame that it can hide quite well beneath the normal overhead of network traffic.

Have we made our point yet? Running nmap as a non-privileged user can unknowingly initiate application sessions, unnecessarily using server resources. If nmap will be used on a production network, always run it as a privileged user!

UDP Ping (`-PU [portlist]`)

A UDP ping is a departure from the other ping types because it uses UDP frames to communicate between devices. Since most of the other ping types focus on TCP frames, the UDP ping can assist in locating active devices through firewalls that the other ping types might not discover.

UDP Ping Operation
The UDP ping attempts to locate a remote device by sending a single UDP frame to the remote device. If an ICMP port unreachable message is returned, a system is alive on the other end:



```
Source          Destination     Summary
--------------------------------------------------------------------
[192.168.0.5] [192.168.0.3] UDP: D=31338 S=42560  LEN=8
[192.168.0.3] [192.168.0.5] ICMP: Destination unreachable (Port
unreachable)
```

If no response is received, it's assumed that the remote system is unavailable. This could be an incorrect assumption if an open UDP port is probed, because many UDP applications don't send a response to any random incoming frame. If possible, the UDP ping should be sent to a port number that is presumed to be closed.

This ping will only run as a privileged user. If this ping is run as a non-privileged user, this amusing message is displayed:

```
Sorry, UDP Ping (-PU) only works if you are root (because
we need to read raw responses off the wire) and only for
IPv4 (cause fyodor is too lazy right now to add IPv6
support and nobody has sent a patch)
```

The UDP ping can specify individual port numbers delimited with commas. If a port is not specified, UDP port 31338 is used by default.

Advantages of the UDP Ping

The UDP ping is simple and innocuous. The single UDP ping frame blends well with other network traffic, and the UDP ping process requires very little network interaction.

Disadvantages of the UDP Ping

The UDP ping relies heavily on ICMP. If ICMP is filtered, there will be no response to the UDP ping.

Not all applications will reply to a random UDP frame that appears on its open port. It's recommended that the UDP ping intentionally use an unavailable port as a destination to ensure a response from the remote station.

The UDP ping only runs as a privileged user, and there are no UDP-based ping alternatives for non-privileged nmap users.

When to use the UDP Ping

The UDP ping is useful when the link between the nmap station and the remote device is not filtered, since packet filters will often drop the ICMP replies. If ICMP can't traverse the network link, this ping type will have limited effectiveness.

It's also important to use UDP ports that are not in use, which can be a challenge when traversing firewalls. If UDP is going to be open through a firewall, it will be to an open service port. However, we don't usually want this ping to use an open UDP port. This limits the effectiveness of this ping through a packet filter or firewall.

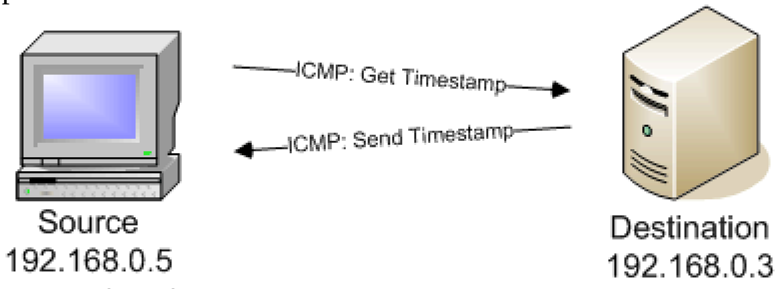
ICMP Timestamp Ping(-PP)

The ICMP timestamp ping is a departure from the usual nmap ping types because it takes advantage of a little-used ICMP message type to determine if a remote station is active. The ICMP timestamp is rarely seen in normal network traffic, but it can be quite useful for determining availability.

An ICMP timestamp ping is designed to allow two separate systems to coordinate their time-of-day clocks. It's not often used, but it's supported across many different TCP/IP stacks and operating systems.

ICMP Timestamp Ping Operation

An ICMP timestamp ping operates similarly to other ICMP-based pings. The source workstation sends an ICMP Get Timestamp message and waits for an ICMP Send Timestamp response.



```
Source         Destination    Summary
-----------------------------------------------------------------------
---------------
[192.168.0.5] [192.168.0.3] ICMP: C Get timestamp
[192.168.0.3] [192.168.0.5] ICMP: R Timestamp
```

If the remote device doesn't respond, the ping fails and the scan does not proceed.

If the nmap is not running as a privileged user, the –PP option provides the following warning:

```
Warning: You are not root -- using TCP pingscan rather than
ICMP
```

The ping process then continues with a TCP connect()-style ping.

Advantages of the ICMP Timestamp Ping
The ICMP timestamp ping doesn't rely on any particular application on the remote device. The ICMP functionality is built into the TCP/IP stack, and many stacks have continued to support this timestamp function.

ICMP is often filtered through firewalls and packet filters. If this ping works, then the network link between the nmap station and the remote device is mostly likely wide open!

Disadvantages of the ICMP Timestamp Ping
Although the ICMP timestamp ping uses little network traffic, the timestamp message is not usually found in normal network conversations. The function itself is esoteric, and although it can provide a time synchronization function for a workstation, most environments rely on Network Time Protocol (NTP) to provide clock synchronization.

The ICMP timestamp ping relies on ICMP, which is often prevented from traversing firewalls or packet filters. This ping is probably not the best choice for scanning through firewalls.

The ICMP timestamp ping will only work for privileged users. Nmap will modify the ping for non-privileged users to use a TCP connect()-like ping. Since a TCP connect() initializes application sessions when accessing an open port, it's important to understand the implications of using this ping type with a non-privileged user!

The `–PP` option ignores any port parameters, so this TCP connect() will always default to port 80. If there's a web server on the other end of the nmap scan, a session will be initialized!

When to use the ICMP Timestamp Ping
The ICMP timestamp ping is useful for non-filtered network connections. If a firewall is in use, this ping option may not provide any successful responses.

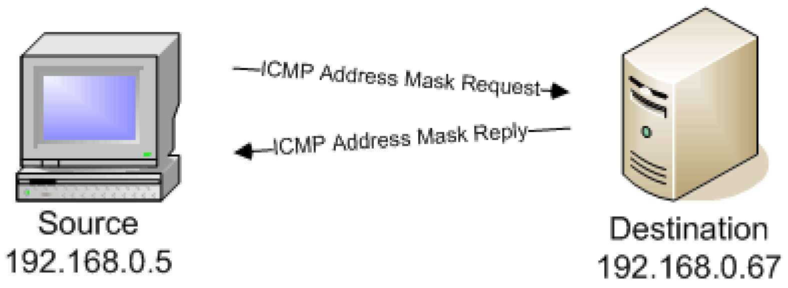
This ping type can only be used by privileged users. If nmap is running as a non-privileged user, this ping type will default to a TCP connect()-like ping.

Non-privileged users have little flexibility when using the `–PP` option. Since this ping type will default to a TCP connect()-like ping method, it's a better option for non-privileged users to use the TCP ACK ping (`-PA`) or TCP SYN ping (`-PS`) so that a port number can be specified.

ICMP Address Mask Ping (`-PM`)

An ICMP address mask request is an antiquated ICMP method that queries an IP gateway for an appropriate subnet mask on the current IP subnet. This ICMP type is extremely rare, and the traffic pattern is very obvious when observing network traces.

ICMP Address Mask Ping Operation
The ICMP address mask ping operates by sending an ICMP address mask request to a remote device. If this device is configured to reply to the ICMP address mask, it will send an ICMP address mask reply:



```
Source          Destination     Summary
-----------------------------------------------------------------------
[192.168.0.5]  [192.168.0.67] ICMP: C Get address mask
[192.168.0.67] [192.168.0.5]  ICMP: R Address mask = [255.255.255.0]
```

If the remote device isn't active or the remote device does not respond to ICMP address mask requests, no response will be seen and the ping will fail.

If the nmap is not running as a privileged user, the `–PM` option provides the following warning:

```
Warning: You are not root -- using TCP pingscan rather than
ICMP
```

The ping process then continues with a TCP connect()-style ping.

Again, with the connect()-style ping! Watch out for this if you're scanning web servers!

Advantages of the ICMP Address Mask Ping A successful ICMP address mask ping can be indicative of an older or unprotected TCP/IP stack or gateway. Most modern operating systems and routers will not respond to this request, or (at the very least) they will not respond to this request from systems that are not on the same subnet. This ping could be useful as a filtering mechanism, since it would identify all systems on the network that have older or unusually open TCP/IP protocol stacks.

ICMP doesn't rely on any particular networking service or application. It's common for ICMP to respond to a request without any particular open or available ports on a system.

Disadvantages of the ICMP Address Mask Ping
An ICMP address mask request is an unusual frame, and it's rarely seen in normal network traffic. When looking at network trace files, the ICMP frames requesting address masks are very obvious.

This ICMP ping type doesn't work on most modern systems, which means that this ping will often fail. If it's important to find active systems, this method won't provide a high percentage of successful pings.

This ping type won't work at all unless the nmap user is privileged. If the nmap user isn't privileged, the ping type will change to a TCP connect()-style ping. Although there is a warning when this occurs, there's no option to stop the scan. Since this ping type doesn't accept a port number variable, this change to a TCP connect()-style ping will only run on the default port of 80. If there's an active web server on the destination station, this uncontrolled ping change will result in the initialization of an application session on the remote device.

ICMP is a difficult protocol to transmit through firewalls and packet filters. Since ICMP is often filtered, this ping has a low percentage of operation through firewalls.

When to use the ICMP Address Mask Ping The ICMP address mask ping is useful on networks that contain older operating systems or gateways.

The successful ping trace shown above was performed against a system using an older version of the VxWorks operating system.

This address mask ping is only useful on networks that allow for the free flow of ICMP frames. If the link contains firewalls or packet filters, a better choice would be a non-ICMP-based ping type.

If the nmap user is non-privileged, this ping type will revert to a TCP connect()-style ping. Since this ping type doesn't allow port specifications, a better ping choice for non-privileged users would be the TCP ACK ping (`-PA`) or the TCP SYN ping (`-PS`).

Don't Ping Before Scanning (`-P0`)

The `–P0` option completely removes the nmap ping requirement from the pre-scanning process. Nmap will still attempt a reverse DNS on the remote station unless the disable reverse DNS (`-n`) option is used.

The `–PN` and `–PD` options are undocumented aliases for the better known –P0 option.

Advantages of the Don't Ping Before Scanning Option Removing the ping process provides an additional level of stealth. The ping is often the first frame sent from the nmap station to the remote device, and removing this conversation minimizes the network traffic between the stations.

The ping process is unnecessary if the end station is known to be active on the network. When the end station's status is already known, using the `–P0` option can remove unnecessary network traffic from the scanning process.

Disadvantages of the Don't Ping Before Scanning Option
If the existence of the remote device isn't known, using the `–P0` option could result in a scan to an IP address that isn't active on the network. If multiple stations are part of the nmap scan, this would create delays while the scan process attempts to contact remote systems that don't actually exist!

When to use the Don't Ping Before Scanning Option
If the end device IP address is already known, then a ping isn't entirely necessary. However, nmap does gather some important timing information from the ping process, so disabling the ping process will put nmap at a disadvantage when the scan begins.

If the nmap ping process fails but the station is known to be on the network, this option can be used to circumvent the nmap ping requirement. This can occur if there is a firewall or packet filter between the nmap station and the remote device that is blocking the nmap pings.

Require Reverse DNS (`-R`)

The require reverse DNS option (`-R`) is nmap's default operation just after the ping and just prior to the scanning process. If correlating a name with an IP address is important, using the `–R` option will ensure a DNS lookup regardless of the scanning type.

Advantages of the Require Reverse DNS Option
When scanning many devices, requiring a reverse DNS will ensure that the scan will attempt to correlate an IP address to a name. Even when the name of a server is known, its IP address may resolve to a completely different name. This resolution process is essential when the "real" name of the device needs to be referenced after a scan is

complete.

Disadvantages of the Require Reverse DNS Option
The `–R` option maps a name to an IP address, but this capability isn't always guaranteed. Not all stations can be resolved through the name resolution process, and not all environments have integrated every workstation into a name resolution system.

The DNS resolution process is relatively slow when compared to most network-related processes. Some DNS clients can add one-half to two seconds of delay to a scan per IP address! When many different systems are scanned simultaneously, this can add a significant amount of delay to the total scan time.

When logging is enabled on the DNS server, every query made to the DNS server will be recorded. If the `–R` option is selected every scanned IP address will appear as a query in the DNS log.

When to use the Require Reverse DNS Option
The reverse DNS option is helpful when station names are an important part of an nmap log. Many organizations use dynamic IP addressing, so an IP address in use during an nmap scan might be used by a completely different station at a later date. By integrating the nmap scan into the name resolution system, a complete and correct name can be associated with an IP address.

If the reverse DNS option is used, there will be delays during the name resolution process. If there are many name resolutions that must occur, it's not unusual to have delays each time a name is resolved.

The name resolution process is often logged on the DNS server, and it's very apparent when examining a network trace file. This type of request includes the device names that are queried, and this makes it stand out when examining a trace file.

Disable Reverse DNS (`-n`)

The disable reverse DNS option (`-n`) is the opposite of the require reverse DNS option (`-R`). If the disable reverse DNS option is selected, nmap will not perform a reverse DNS lookup on the destination IP address. Disabling the DNS process eliminates the name resolution process, which is one of the more time-consuming aspects of a workstation scan.

Advantages of the Disable Reverse DNS Option
DNS lookups are relatively slow when compared with the scanning process. Name resolution can be very inefficient, so removing this requirement can provide some significant time savings.

A name resolution process doesn't guarantee that a name will actually be resolved. Individual workstations often aren't included on the DNS server, so the resulting name lookup will not return any name that can be correlated with an IP address. There's no reason to have a name resolution process if no name will be obtained!

If the name lookup doesn't occur, there's no corresponding entry in the DNS logs. If it's important to conserve resources on the DNS server, this scan option will avoid sending queries to the DNS server.

Disadvantages of the Disable Reverse DNS Option
The reverse DNS process can associate a system name with an IP address. In environments where the IP addresses are assigned dynamically, a station's IP address can change from one day to the next. If the reverse DNS option is disabled, the station name is not queried and the name will not appear in the nmap scan. In dynamic environments, querying the name during the scan may be the only opportunity to accurately correlate an IP address with a name!

When to use the Disable Reverse DNS Option
If the nmap scan needs to keep a low profile on the network, it may be prudent to disable any name queries. Many DNS servers log name resolutions, so running an nmap scan without disabling name resolution may cause the nmap station to appear in the DNS log as it attempts to resolve the name of every workstation it scans!

Scanning many different devices will result in a large number of queries to a DNS server. The scan described above queried the DNS server over 250 times in two minutes! If the name of a device isn't important, using the `-n` option will save CPU cycles on both the nmap client and the DNS server.

DNS queries are notoriously slow. Using the disable reverse DNS option can increase scan time significantly if multiple stations are scanned.

Chapter 4:

Operating System Fingerprinting

The operating system fingerprinting process is one of nmap's most powerful features. The fingerprinting process is impressive in its scope, and its capabilities are some of the most unique in the industry.

The OS fingerprinting process is not a port scan, although it works in conjunction with nmap's scanning processes. Nmap's operating system fingerprinting is based on a remote device's responses when it's sent a group of very specific packets. There will be subtle differences in the responses received from different operating systems under certain circumstances. If a particular operating system receives a TCP ACK frame to a closed port, it may react differently than other operating systems receiving the same frame. It's these minor response variations that allow nmap to build detailed "fingerprints" for different operating systems and devices.

Fyodor discusses the history and details of these packet types in his extensive paper describing the OS fingerprinting process. His research and experiences with these fingerprinting methods can be found at http://www.insecure.org/nmap/nmap-fingerprinting-article.html. Fyodor discusses many different techniques for harvesting information from a remote device in this article, including methods that weren't included with nmap.

The OS fingerprinting process is not a version detection scan (-sV), although many methodologies between the two processes are similar. For example, both the version scan and the OS fingerprinting scan rely on the nmap scanning process to identify active devices and their available ports. However, The OS fingerprinting process uses techniques not found in version detection, such as a standard method of operating system probing and a modular operating system definition file.

Operating System Fingerprinting (`-O`) Operation Before the operating system fingerprinting process begins, nmap performs a normal ping and scan. During the nmap scan, nmap determines device availability and categorizes the ports on the remote device as open, closed, or filtered.

This list of port dispositions is important because the operating system fingerprinting process needs to query both open ports and closed ports to obtain accurate operating system fingerprints.

Once the open and closed ports are identified, nmap begins the OS fingerprinting procedure. The OS fingerprinting process consists of an operating system probe, followed by series of TCP handshakes that are used for testing responses to the TCP uptime measurement options, TCP sequence predictabilities, and IP identification sequence generation.

These simple tests are designed to gather extensive information about an operating system while using a minimum of network traffic.

It's extraordinary how much operating system information can be gathered from a device without authenticating or starting an application session. If a device is on the network with an IP address, some extremely simple and noninvasive tests will uncover amazing amounts of information!

A normal OS fingerprinting process will uncover the following information:
```
Device type: general purpose
Running: Microsoft Windows NT/2K/XP
OS details: Microsoft Windows XP SP2
TCP Sequence Prediction: Class=truly random
                         Difficulty=9999999 (Good luck!)
IPID Sequence Generation: Incremental
```

The `nmap-os-fingerprints` Support File

The `nmap-os-fingerprints` support file contains a definition of every operating system fingerprint that nmap recognizes. As new operating system fingerprints are created and released, this text file is simply updated with the new fingerprint definitions.

This is the definition for a Microsoft Windows XP SP2 operating system from the `nmap-os-fingerprints` file:

Fingerprint Microsoft Windows XP SP2

```
Class Microsoft | Windows | NT/2K/XP | general purpose
TSeq(Class=TR%gcd=<6%IPID=I)
T1(DF=Y%W=6360|805C|FFAF%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=6360|805C|FFAF%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=B0%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
```

Each operating system definition contains similarly formatted information. Each line contains information that contributes to the overall operating system fingerprint.

All of these attributes are documented below. This is the most comprehensive documentation of the operating system fingerprinting process available anywhere!

`nmap-os-fingerprints`: Fingerprint

```
Fingerprint Microsoft Windows XP SP2
```

The first line in a fingerprint definition is labeled `Fingerprint`. This line references the name of the operating system fingerprint, and this information is displayed on the "OS Details:" entry on the nmap output:

```
OS details: Microsoft Windows XP SP2
```

`nmap-os-fingerprints`: Class

```
Class Microsoft | Windows | NT/2K/XP | general purpose
```

The `Class` line is a combination of four different variables. The structure of this line corresponds to:

```
Manufacturer | OS Name | Version | Device Type
```

If one of these variables is unknown or does not apply, the variable is left blank. The first three variables are combined on the "Running:" line of the nmap output, and the last variable is used on the "Device type:" output line:

```
Device type: general purpose
Running: Microsoft Windows NT/2K/XP
```

`nmap-os-fingerprints`: TSeq

```
TSeq(Class=TR%gcd=<6%IPID=I)
```

The `TSeq` line contains the fingerprint information for TCP Sequence Prediction. This is

the fingerprint that nmap uses to determine if initial sequence numbers (ISNs) can be predicted based on past results:
```
TCP Sequence Prediction: Class=64K rule
                         Difficulty=1 (Trivial joke)
```
or
```
TCP Sequence Prediction: Class=truly random
                         Difficulty=9999999 (Good luck!)
```

Nmap clearly identifies when a TCP initial sequence number would be impossible to predict!

The Importance of TCP Sequence Prediction Analysis
If the TCP sequences of a remote device are understood, then that remote device is more susceptible to malicious activity such as TCP hijacking. TCP hijacking is a technique that allows a third-party to "interrupt" an existing TCP connection between two devices. The malicious third party can then masquerade as one of the original stations, allowing them to send unwanted information to the other device. A major technical aspect of the hijacking process is the ability of the attacking station to predict the TCP sequence numbers.

TSeq: The `Class` Attribute
The TSeq `Class` attribute refers to the predictability of a remote device's TCP initial sequence number (ISN). Many of these class attributes also include more detailed attributes to assist nmap in matching a fingerprint.

- `Class=C`
  Amazingly, this fingerprint attribute describes an ISN that is the same value with every SYN/ACK. This class refers to a constant (`C`) number, and it will often include a `Val` attribute that represents the constant ISN value that is always seen from the remote device.

- `Class=64K`
  A fingerprint with `Class=64K` describes devices that have an initial sequence number that increases by 64,000 with each SYN/ACK.

- `Class=i800`
  These `i800` devices have ISNs that increase by a fixed increment of 800 with each SYN/ACK.

- `Class=TD`
  Some systems, including Windows-based devices, increase the initial sequence number by a fixed amount during a specific time period. This time dependent

(`TD`) model often includes fingerprint attributes based on the greatest common divisor (`gcd`) and a sequence index (`SI`) of the initial sequence number.

- `Class=RI`
  The random increments (`RI`) class describes a series of ISNs that increment, but there is no method of predicting the sequence number. This class often includes fingerprint attributes based on the greatest common divisor (`gcd`) and a sequence index (`SI`) of the initial sequence number.

- `Class=TR`
  If the initial sequence numbers are shown to be completely random, they are fingerprinted with the truly random (`TR`) class.

These `Class` attributes can also be combined together to show that a remote device might have initial sequence numbers that are truly random, random increments, or fixed increments of 800. This would be shown in the `nmap-os-fingerprints` file as `Class=TR|RI|i800`.

TSeq: The `IPID` Attribute

The `IPID` attribute refers to the IP identification bytes in the IP header. These values provide important information, since the IPID can be used for non-standard purposes. Nmap's idlescan is an example of how a predictable IPID can have unintended uses.

- `IPID=C`
  In rare instances, some systems provide IPIDs that are always a constant (`C`) number.

- `IPID=I`
  A system matching the Incremental (`I`) fingerprint increases the IPID by a standard increment with each sent packet.

- `IPID=BI`
  The broken incremental (`BI`) fingerprint refers to a system (usually Windows-based) that increases by 256 each time a packet is sent. This is probably caused by an unintentional error in Microsoft's IP stack, but it's still a predictable error.

- `IPID=RPI`
  The random positive integral (`RPI`) fingerprint is based on an IPID that increases each time a packet is sent, but the increase is by an apparently random amount.

- `IPID=RD`
  Random distributions (`RD`) are fingerprint references that identify IPIDs that increase or decrease randomly each time a packet is sent.

- `IPID=Z`
  In some cases, the IPID will always be a zero (`Z`) value.

TSeq: Timestamp Option Sequencing
The TCP timestamp option is a standard method of calculating round-trip time between stations, documented in RFC 1323. However, this exact change in timestamp values will vary between operating systems. These variances can be identified and fingerprinted with the timestamp attributes.

Nmap looks for the change in timestamp each second and attempts to categorize it into five separate groups:

- `TS=0`
  If the returned timestamp is zero, nmap categorizes it as a zero () TCP timestamp sequence.

- `TS=2HZ`
  A timestamp sequence that increases twice in one second is defined as `2HZ`. Nmap uses the abbreviation for hertz, `HZ`, to reference the number of frequencies per cycle. In this case, `HZ` refers to the number of timestamps incremented per second.

- `TS=100HZ`
  If a timestamp increases by 100 every second, it's assigned a `TS` reference of 100HZ.

- `TS=1000HZ`
  An increase of one thousand timestamps per second is categorized as `1000HZ`.

- `TS=U`
  If any remote device does not return a timestamp, it's fingerprinted as an unsupported system (`U`).

`nmap-os-fingerprints`: Test 1 (`T1`) through Test 7 (`T7`)

```
T1(DF=Y%W=6360|805C|FFAF%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=6360|805C|FFAF%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
```

Test 1 (abbreviated in the fingerprint as `T1`) through Test 7 (`T7`) refer to the fingerprints that result from seven frames sent to the remote device. Each test has a specific function, and the results of each test are correlated with the `nmap-os-fingerprints` file to match with the target station's operating system. The tests are very specific, and the packets received in reply are scrutinized for their identifiable response patterns.

Prior to running test 1 through test 7, nmap chooses an open port and a closed port to use for the appropriate tests. If only one of these two options is available, nmap provides a message regarding its ability to accurately fingerprint:

```
Warning:  OS detection will be MUCH less reliable because
we did not find at least 1 open and 1 closed TCP port
```

This is a description of the tests and a summary of the frame that is sent to the target:

T1: Test 1 sends a SYN frame with a mix of TCP options to an open port. These options consist of a window scale option of 10, a maximum segment size of 265, and a timestamp value of 1061109567.

T2: Test 2 sends a NULL TCP frame (no flags set) to an open port. This frame includes the same TCP options as those in Test 1.

T3: Test 3 sends a TCP frame with the SYN, FIN, PSH, and URG flags to an open port. This frame also includes the same TCP options as those found in test 1 and test 2.

T4: Test 4 sends a TCP ACK to the open port.

T5: Test 5 begins the fingerprint tests to the previously found closed port. This test sends a TCP SYN to the closed port.

T6: Test 6 sends a TCP ACK to the closed port.

T7: Test 7 sends a TCP frame with the FIN, PSH, and URG flags to the closed port.

The `T1` to `T7` Attributes

The seven test fingerprints all follow the same syntax, although it's not a requirement for every test line to include every possible attribute. Each test fingerprint follows the same attribute order:

`Resp:` The fingerprint shows a `Y` if a response is received, and it displays an `N` if a response is not received.

`DF:` This fingerprint attribute specifies whether the "Don't Fragment" bit is set in the response frame.

`W:` The fingerprint displays the window size or sizes (separated by "|") expected in the response frame.

`ACK:` This attribute refers to the expected ACK value. This can display an `S` to mean the same sequence number that was sent in the test frame, or `S++` if the reply includes the initial sequence's ACK number plus one. If an `O` is displayed, some other value was returned from the test frame.

`Flags:` The flags attribute displays the TCP flags that are enabled in the reply, in this order:

`B` = Bogus (the flag in the reply frame isn't a real TCP flag)
`U` = Urgent
`A` = Acknowledgement
`P` = Push
`R` = Reset
`S` = Synchronize
`F` = Final

`Ops:` The `Ops` attribute displays the TCP options that are enabled in the reply:

`L` = End of List
`N` = No Op
`M` = Maximum Segment Size
`E` = Echoed
`W` = Window Scale
`T` = Timestamp

`nmap-os-fingerprints`: The Port Unreachable Test (`PU`)

```
PU(DF=N%TOS=0%IPLEN=B0%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=1
34%DAT=E)
```

For the port unreachable test, nmap selects a closed port and sends a UDP frame. The ensuing ICMP port unreachable reply includes an "echo" of data from the originating packet. The ICMP port unreachable fingerprint compares these attributes:

`DF` = If the don't fragment bit is set, the fingerprint shows a `Y`. If the don't fragment bit is not set, the fingerprint shows an `N`.

`TOS` = The IP header's type of service (`TOS`) byte is compared to this TOS fingerprint. The type of service fingerprint is shown as a hexadecimal value.

`IPLEN` = The fingerprint displays the IP datagram total length (`IPLEN`) in bytes from the response packet's IP header, referenced as a hexadecimal value.

`RIPTL` = The IP datagram total length (in hex) repeated back (`RIPTL`) to the nmap station is listed as this attribute. This is the IP datagram total length from the echoed IP header, not the bytes in the "real" IP header of the frame. The value of the "real" IP header is referenced in the `IPLEN` attribute.

`RID` = If the IP identification bytes in the reply (`RID`) are identical to the original frame, the fingerprint contains an `E`. If the echoed IPID doesn't match the original, the RID fingerprint attribute contains an `F`.

`RIPCK` = This fingerprint attribute is the comparison of the returned IP checksum (`RIPCK`) in the echoed packet with the checksum in the original packet. An `E` signifies a match, while an `F` signifies that the two checksums are different.

`UCK` = The UDP checksum (`UCK`) fingerprint attribute compares the UDP checksum in the original frame with the echoed data in the response frame. An `E` signifies a match, while an `F` signifies that the two checksums are different.

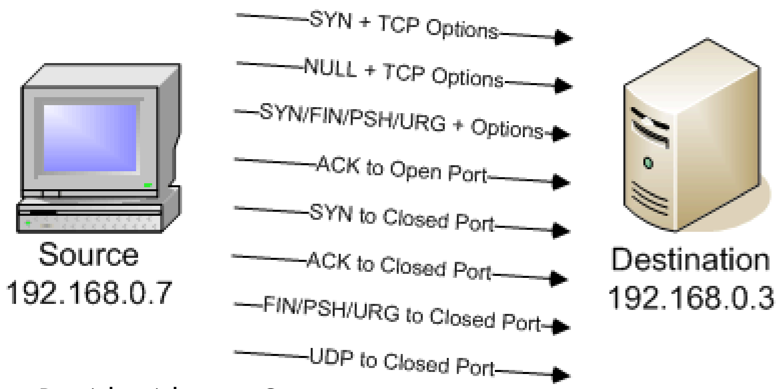
`ULEN` = The UDP length (`ULEN`) in the response frame echo should match the value from the original frame. Nmap sends 0x134 bytes in the original frame, and the fingerprint displays the value in hex that should appear in the response frame echo.

`DAT` = If the data (`DAT`) in the returned packet matches that of the original frame, this fingerprint assigns an `E` to this attribute. If the data is not identical, an `F` is assigned.

The Operating System Fingerprinting Process

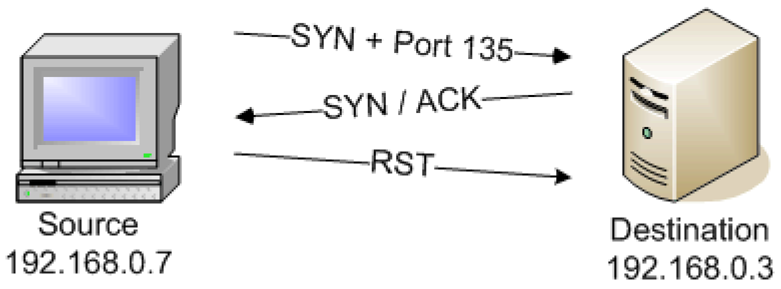
The operating system fingerprinting process takes place in a different order than what's shown in the structure of the fingerprint!

The operating system fingerprinting probes begin with Test 1 through Test 7, followed immediately by the UDP-based ICMP port unreachable test. The responses to these probes are compared to the T1-T7 fingerprints in the hopes of locating some likely matches.



```
Source         Destination    Summary
-----------------------------------------------------------------------
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56414 SYN SEQ=3375903318 LEN=0
WIN=2048
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56415     WIN=2048
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56416 SYN FIN URG
SEQ=3375903318 LEN=0 WIN=2048
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56417     ACK=0 WIN=4096
[192.168.0.7] [192.168.0.3] TCP: D=1 S=56418 SYN SEQ=3375903318 LEN=0
WIN=2048
[192.168.0.7] [192.168.0.3] TCP: D=1 S=56419     ACK=0 WIN=4096
[192.168.0.7] [192.168.0.3] TCP: D=1 S=56420 FIN URG SEQ=3375903318
LEN=0 WIN=4096
[192.168.0.3] [192.168.0.7] TCP: D=56414 S=135 SYN ACK=3375903319
SEQ=1982576985 LEN=0 WIN=65535
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56414 RST WIN<<10=0
[192.168.0.3] [192.168.0.7] TCP: D=56415 S=135 RST ACK=3375903318 WIN=0
[192.168.0.7] [192.168.0.3] UDP: D=1 S=56407  LEN=308
[192.168.0.3] [192.168.0.7] TCP: D=56416 S=135 SYN ACK=3375903319
SEQ=3856135382 LEN=0 WIN=65535
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56416 RST WIN<<10=0
[192.168.0.3] [192.168.0.7] TCP: D=56417 S=135 RST WIN=0
[192.168.0.3] [192.168.0.7] TCP: D=56418 S=1 RST ACK=3375903319 WIN=0
[192.168.0.3] [192.168.0.7] TCP: D=56419 S=1 RST WIN=0
[192.168.0.3] [192.168.0.7] TCP: D=56420 S=1 RST ACK=3375903319 WIN=0
[192.168.0.3] [192.168.0.7] ICMP: Destination unreachable (Port
unreachable)
```

Nmap then performs six TCP SYN scans to the open port. The resulting SYN/ACK responses are used to compare TCP initial sequence numbers, IP identification values, and TCP timestamp option sequences.



```
Source         Destination    Summary
-----------------------------------------------------------------------
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56408 SYN SEQ=3375903319 LEN=0
WIN=3072
[192.168.0.3] [192.168.0.7] TCP: D=56408 S=135 SYN ACK=3375903320
SEQ=1634073962 LEN=0 WIN=65535
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56408 RST WIN<<10=0
```

```
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56409 SYN SEQ=3375903320 LEN=0
WIN=1024
[192.168.0.3] [192.168.0.7] TCP: D=56409 S=135 SYN ACK=3375903321
SEQ=1378387651 LEN=0 WIN=65535
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56409 RST WIN<<10=0
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56410 SYN SEQ=3375903321 LEN=0
WIN=2048
[192.168.0.3] [192.168.0.7] TCP: D=56410 S=135 SYN ACK=3375903322
SEQ=2771984018 LEN=0 WIN=65535
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56410 RST WIN<<10=0
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56411 SYN SEQ=3375903322 LEN=0
WIN=2048
[192.168.0.3] [192.168.0.7] TCP: D=56411 S=135 SYN ACK=3375903323
SEQ=1187915667 LEN=0 WIN=65535
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56411 RST WIN<<10=0
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56412 SYN SEQ=3375903323 LEN=0
WIN=3072
[192.168.0.3] [192.168.0.7] TCP: D=56412 S=135 SYN ACK=3375903324
SEQ=2260369583 LEN=0 WIN=65535
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56412 RST WIN<<10=0
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56413 SYN SEQ=3375903324 LEN=0
WIN=3072
[192.168.0.3] [192.168.0.7] TCP: D=56413 S=135 SYN ACK=3375903325
SEQ=212338365 LEN=0 WIN=65535
[192.168.0.7] [192.168.0.3] TCP: D=135 S=56413 RST WIN<<10=0
```

Once these probes are complete, nmap has the information it needs to compare to the `nmap-os-fingerprints file`. If there's a match, nmap will display the operating system in the nmap output. If there are multiple matches, nmap provides a message informing of the multiple matches:

```
Too many fingerprints match this host to give specific OS
details
```

If the operating system fingerprinting didn't find any matches, this message is displayed:

```
No OS matches for host (If you know what OS is running on
it, see
http://www.insecure.org/cgi-bin/nmap-submit.cgi).
```

This web page contains an nmap fingerprint submission form for contributions to the `nmap-os-fingerprints` file. This page requests OS and classification information about the device, including an IP address so Fyodor can scan the device for additional fingerprint testing.

The operating system fingerprinting option needs to run as a privileged user. Otherwise this totally bogus message will appear, dude:

```
TCP/IP fingerprinting (for OS scan) requires root
privileges which you do not appear to possess. Sorry, dude.
```

Advantages of Operating System Fingerprinting

The operating system fingerprinting process provides detailed information about the operating system running on a device. In some cases, the exact version number of the operating system and detailed hardware information can be determined with the OS fingerprinting option.

Some organizations have policies forbidding certain operating systems from attaching to the network. The OS fingerprinting option can assist with locating systems that are out of compliance, and can also provide information about the operating system running on the "rogue" station. When this option is combined with the version scan (`-sV`), specific services can also be checked for compliancy.

The OS fingerprinting process is a simple set of queries, and most of the frames are relatively harmless. It's amazing the level of the detail that can be determined based on the nuances of simple packet responses from another device. This process never opens an application session, which makes the results even more amazing!

Disadvantages of Operating System Fingerprinting

The OS fingerprinting process requires privileged user access. This scan will not run if a non-privileged user attempts to use the `–O` option.

Although there are only about thirty frames that traverse the network during an OS fingerprinting process, some of the frames used to query the remote device are frame types that would never occur on a normal network. For example, it's unusual to see a frame with the SYN, FIN, PSH, and URG flags that would also include numerous TCP options. A trained eye will quickly identify these unusual frames, assuming that someone is watching the network during that timeframe.

When to use Operating System Fingerprinting

The operating system fingerprinting option is often integrated into many organization's compliance checks. If an outdated or unexpected operating system is seen on the network, the security group can follow their policies to identify and remove the noncompliant station from the network.

In some cases, a particular operating system may have known vulnerabilities that need to be patched. The OS fingerprinting process can assist with locating all of the specific operating system versions on the network, ensuring that organization's vulnerable holes will be patched.

If nmap can identify this level of operating system detail without ever launching an application session or authenticating, then anyone else on the network can obtain the same information! The OS fingerprinting process can help the security team understand what everyone else can see, which will assist in making the network and firewall infrastructure even more secure.

Limit Operating System Scanning (`--osscan_limit`)

The operating system fingerprinting process is most accurate when both open ports and closed ports are available for testing. If only one type of port is available, the fingerprinting process will not be as precise. In this situation, nmap provides a warning message:

```
Warning:  OS detection will be MUCH less reliable because
we did not find at least 1 open and 1 closed TCP port
```

The fingerprinting process will still function, but the results will not be optimal.

If the fingerprinting process needs to be as accurate as possible, the `--osscan_limit` option will abort OS fingerprinting if both open and closed ports aren't available. This will ensure that OS fingerprinting will run only if the conditions are perfect. This also saves time if a remote device is identified but the port disposition is in question because of firewalls or packet throttling on the remote device.

If many devices will be scanned, this option can save a lot of time!

More Guessing Flexibility (`--osscan_guess`, `--fuzzy`)
The `--osscan_guess` option is relatively unknown, but that's probably because it's not well documented. A section in the CHANGELOG refers to the option as "secret." The `--fuzzy` option is an alias for `--osscan_guess`.

The `--osscan_guess` option forces nmap to "guess" when operating system fingerprinting can't find a perfect match. Occasionally, nmap will decide to invoke this option automatically if certain parameters are met.

Additional, Advanced, and Aggressive (`-A`)
The aptly named Additional, Advanced, and Aggressive option (`-A`) is a shortcut for running both the operating system fingerprinting process (`-O`) and the version scanning process (`-sV`) during the same nmap scan. This shortcut still requires a port scan to locate open and closed ports, but there's only one abbreviation to remember if both options are required.

The `–A` option only adds the operating system fingerprinting and version detection options. It doesn't change or add any scan types or option settings.

According to the nmap man page, Fyodor reserves the right to expand on this option in the future. Currently, this option is intended to be time saver, and nothing more.

CHAPTER 5:

HOST AND PORT OPTIONS

Nmap includes many options related to the IP addresses and port numbers that will be used during the scan. These option parameters range from excluding particular IP addresses from the scan to creating fake decoys that traverse the network! With these many host and port options, nmap can be customized to create the perfect scanning environment.

Exclude Targets (`--exclude <host1 [,host2] [,host3]...>`)
The `--exclude` option provides nmap with a list of IP addresses that will be avoided when performing a scan. This is useful for performing a scan on a subnet of devices while avoiding the scanning of routers or production servers. For example, the command:
```
nmap –sS 192.168.0.1/24 –-exclude 192.168.0.2-4, 192.168.0.77
```
will run nmap's TCP SYN scan (`-sS`) to the range of IP addresses at 192.168.0.0 through 192.168.0.255 (`192.168.0.1/24`), while excluding the devices at 192.168.0.2, 192.168.0.3, 192.168.0.4, and 192.168.0.77.

The list of IP addresses associated with the `--exclude` option should be separated by commas. The excluded hosts can be identified with wildcards, subnet values, or any other standard nmap reference. Refer to the "Nmap Target Specifications" section for more information on IP address specifications.

The `--exclude` option cannot be used in conjunction with the `--excludefile` option. It's one or the other, but not both.

The nmap 3.81 man page shows a quotation symbol (`"`) at the end of the hostname elements description. Quotation marks are usually used in pairs on the command line to prevent special character interpretation, but this isn't the case with the `--exclude` option. These quotation marks are probably an unintended typographical error.

Exclude Targets in File (`--excludefile <exclude_file>`)
The `--excludefile` option is similar to the `--exclude` option, except the addresses to be excluded are listed in a file. The target address exclusions should be listed with one address per line, and the exclusion lines cannot be separated with spaces or tabs. Unlike the "read targets from file" option (`-iL`), the `--excludefile` option will only accept one exclusion parameter per line.

Multiple IP addresses can be defined on each line using nmap's host specification parameters. For example, the following IP address specification will exclude all hosts between 192.168.0.1 and 192.168.0.7, 192.168.0.10, and all hosts between 192.168.1.0 and 192.168.1.255.
```
192.168.0.1-7
192.168.0.10
192.168.1.*
```

The exclusion file can include IP addresses that aren't necessarily part of the current scan. Therefore, a single exclude file could be used for all nmap scans regardless of the IP addresses on the nmap command line. If there are certain systems that should never be scanned (DNS servers, file servers, telephone switches, etc.), they can be added to a 'permanent' exclude file that can be used for all nmap scans.

The `--excludefile` option cannot be used on the same nmap command line as the `--exclude` option. Only one of these options can provide the primary exclusion list for the nmap scan.

Read Targets from File (`-iL <inputfilename>`)
The `–iL` option reads target IP address from a specified filename. If the target addresses are piped into nmap from another process, the single hypen (`-`) can be used to specify the standard input (`stdin`) instead of a filename.

Because of the flexibility required to receive input from another process, the `–iL` option is more adaptable than nmap's exclusion options. Included addresses can be separated by tabs, spaces, or by separate lines.

When this option is used, the filename is the only valid input. If any additional host addresses are included on the command line, they will be ignored without any warning message.

If the host exclusion options (`--exclude` or `--excludefile`) are used in conjunction with the `–iL` option, the excluded addresses will override any inclusions on the command line or file.

Pick Random Numbers for Targets (`-iR <num hosts>`)
This option chooses completely random destination addresses as the nmap scan input. These random values do not consider any of the filename options for including or excluding host addresses (`-iL` and `--excludefile`, respectively), and the `--exclude` option is also ignored. If a series of target addresses needs to be randomly scanned, the `--randomize_hosts` option should be used in conjunction with a series of include and exclude options instead of using the `–iR` option.

The `–iR` option requires a quantity of random IP addresses that will be used for this nmap scan. If a quantity of hosts is not specified, the nmap scan aborts with this message:
```
ERROR: -iR argument must be the maximum number of random IPs you wish
to scan (use 0 for unlimited)
QUITTING!
```
As this message shows, specifying zero (0) hosts will run an unlimited number of random addresses through the nmap scan. The nmap man page has a good example of using this feature to find a random web server:
```
# nmap –sS –PS80 –iR 0 –p 80
```
This example runs a TCP SYN scan (`-sS`) using a SYN ping on port 80 (`-PS80`) to an unlimited number of random IP addresses (`-iR 0`). The SYN scan only scans port 80 (`-p 80`).

This scan won't begin to report any results until 500 hosts are identified. The `--debug` or `--packet_trace` options can be used to watch nmap working, but take care not to miss the IP address listing when nmap hits the 500 mark! It may be helpful to run this type of scan in conjunction with one of nmap's logging options.

Randomize Hosts (`--randomize_hosts, -rH`)

The `--randomize_hosts` option rearranges the group of hosts in an nmap scan. Groups of 2,048 hosts at a time are randomly chosen, making the entire scan less conspicuous when examining traffic patterns.

The `-rH` abbreviation is an undocumented alias for the `--randomize_hosts` option.

No Random Ports

When nmap performs a port scan, it automatically randomizes the destination port numbers to make the scans blend easier with the normal network traffic patterns. For example, this excerpt of a default TCP SYN scan shows a random destination TCP port distribution:

```
Source        Destination   Summary
-------------------------------------------------------------------------
[192.168.0.7] [192.168.0.1] TCP: D=13713 S=40950 SYN SEQ=1553882748 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.1] TCP: D=1456 S=40950 SYN SEQ=1553882748 LEN=0
WIN=1024
[192.168.0.7] [192.168.0.1] TCP: D=2564 S=40950 SYN SEQ=1553882748 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.1] TCP: D=83 S=40950 SYN SEQ=1553882748 LEN=0
WIN=3072
[192.168.0.7] [192.168.0.1] TCP: D=721 S=40950 SYN SEQ=1553882748 LEN=0
WIN=1024
[192.168.0.7] [192.168.0.1] TCP: D=811 S=40950 SYN SEQ=1553882748 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.1] TCP: D=364 S=40950 SYN SEQ=1553882748 LEN=0
WIN=3072
[192.168.0.7] [192.168.0.1] TCP: D=51 S=40950 SYN SEQ=1553882748 LEN=0
WIN=3072
[192.168.0.7] [192.168.0.1] TCP: D=722 S=40950 SYN SEQ=1553882748 LEN=0
WIN=4096
```

When using the `-r` option, however, the port numbers are scanned sequentially:

```
Source        Destination   Summary
-----------------------------------------------------------------------
[192.168.0.7] [192.168.0.1] TCP: D=1 S=54036 SYN SEQ=2232636309 LEN=0
WIN=2048
[192.168.0.7] [192.168.0.1] TCP: D=2 S=54036 SYN SEQ=2232636309 LEN=0
WIN=3072
[192.168.0.7] [192.168.0.1] TCP: D=3 S=54036 SYN SEQ=2232636309 LEN=0
WIN=2048
[192.168.0.7] [192.168.0.1] TCP: D=4 S=54036 SYN SEQ=2232636309 LEN=0
WIN=1024
```

```
[192.168.0.7] [192.168.0.1] TCP: D=5 S=54036 SYN SEQ=2232636309 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.1] TCP: D=6 S=54036 SYN SEQ=2232636309 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.1] TCP: D=7 S=54036 SYN SEQ=2232636309 LEN=0
WIN=4096
[192.168.0.7] [192.168.0.1] TCP: D=8 S=54036 SYN SEQ=2232636309 LEN=0
WIN=4096
```

The `–r` option also "unrandomizes" the IP protocol scan (`-sO`).

Source Port (`--source_port <portnumber>`, `-g <portnumber>`)

During an nmap scan, the source port is often used to store counters and other information during the course of a scan. The source port doesn't often need to be defined, but the `--source_port` option provides a method of forcing the source port to a specified value if the need arises.

Since nmap uses this source port as an information source during a scan, forcing the value to a specific number will have a detrimental performance impact on nmap's scanning efficiency.

The `–g` abbreviation is an alias for the more descriptive `––source_port` option name.

Specify Protocol or Port Numbers (`-p <port_range>`)

During a scan, nmap defaults to scanning ports 1 through 1,024, as well as any other ports listed in the `nmap-services` support file. This can amount to thousands of ports for a default scan to a single host!

The `–p` option provides a method of specifying the port numbers to be probed during an nmap scan. If the scan is TCP or UDP based, the port numbers can be any range of numbers between 0 and 65,535. For IP protocol scans (`-sO`), the `–p` option refers to IP protocol numbers between 0 and 255.

Since nmap can scan for different protocol types on a single command line, there are additional arguments that can be included to specify TCP or UDP port ranges. TCP port ranges can be specified with the `T:` argument, and UDP ports with the `U:` argument. These additional specifications are only required when a UDP scan is requested (`-sU`) on the same scan as a TCP-based scan type. The UDP scan is the only nmap scan type that can identify UDP ports. If neither protocol type is specified, the port range will apply to both types.

Port ranges can be specified as individual numbers, or ranges of numbers, with each group of numbers separated by commas. For example,

```
# nmap –sS 192.168.0.1 –p 23,80,111-124,155-
```

will perform a TCP SYN scan of IP address 192.168.0.1 on TCP ports 23, 80, 111 through 124, and all TCP ports between 155 and 65,535.

The nmap man page is slightly incorrect when it states that a specification of `60000-` would scan all ports greater than 60,000, since port 60,000 would also be included as part of the scan.

If duplicate port numbers are listed on the command line, nmap will question the user's caffeine intake:

```
WARNING:  Duplicate port number(s) specified.  Are you alert enough to
be using Nmap?  Have some coffee or Jolt(tm).
```

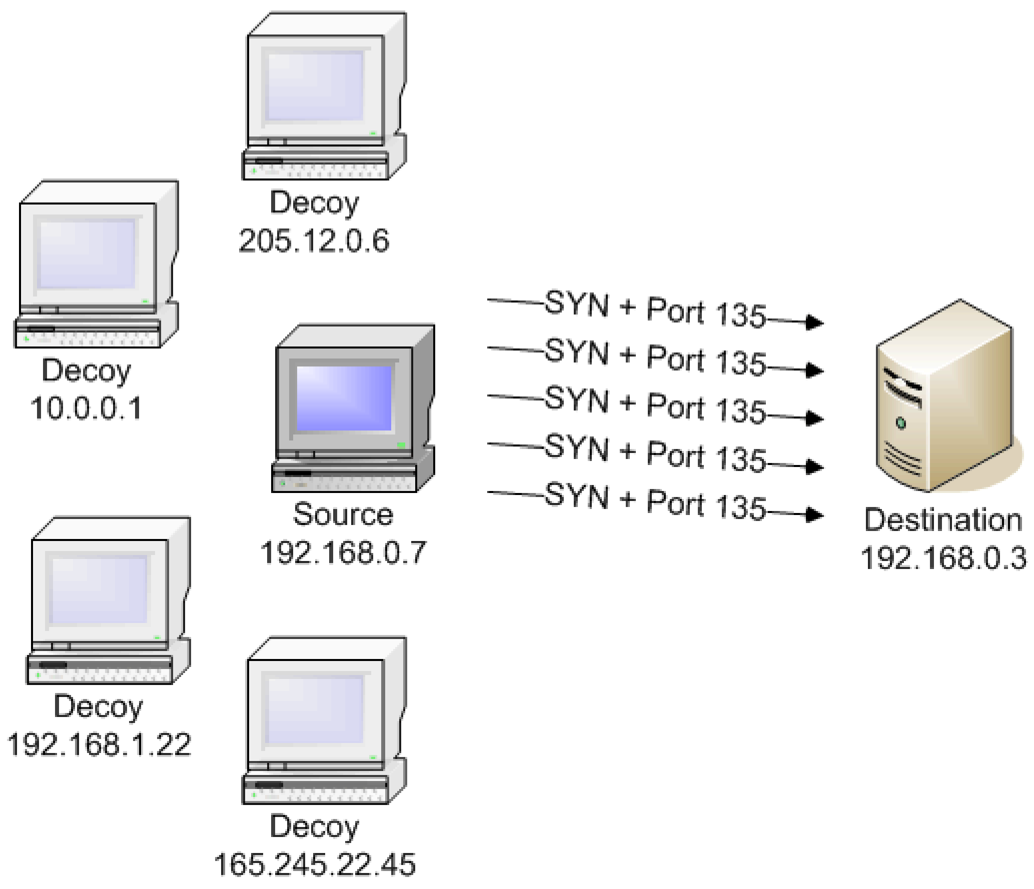
Fast Scan Mode (`-F`)

The "fast scan" nmap option doesn't have the most accurate name, since this scan mode doesn't actually change any of the nmap packet timing policies. Instead, this fast scan mode (`-F`) limits nmap scans to the ports found in the `nmap-services` file. If the nmap is performing an IP protocol scan (`-sO`), the fast scan mode scans only the IP protocol types listed in the `nmap-protocols` support file.

When used with a customized `nmap-services` or `nmap-protocols` support file, this fast scan mode can be an easy method of scanning specific port numbers without using the port specification option (`-p`).

Create Decoys (`-D <decoy1 [,decoy2][,ME],...>`)

The `–D` option allows nmap to create packets that appear to originate from other IP addresses. This IP address spoofing allows nmap to simulate many different devices when performing certain scans. Decoys are useful for testing intrusion detection systems (IDS) or intrusion prevention systems (IPS) and their reaction to multiple simultaneous scans. Some IDS/IPS systems capabilities may be limited, and the decoy scan can assist in determining the maximum amount of visibility expected from these systems under different attack configurations.



The maximum number of decoys (as defined in `nmap.h`) is 128. This is a very high maximum that probably won't be exceeded in normal use. If too many decoys are used during a scan, the performance will decrease as the remote host throttles network traffic or if it is overwhelmed.

Decoys won't work with the RPC scan (`-sR`), Idlescan (`-sI`), TCP connect() scan (`-sT`), or the FTP bounce scan (`-b`). The decoys will appear during the nmap ping process, many of the other nmap scans, and during an operating system scan (`-O`).

One of the `–D` options is to use `ME` as a decoy name. This identifies the nmap station as one of the devices to use during the scan, and the frames will be sent in the order in which the decoys are added to the nmap command line. If `ME` is placed near the end of the list, there's a better chance of the nmap station circumventing IDS or IPS alarms.

This feature is best used when the destination device is not on the same IP subnet as the nmap station. Although the IP addresses are spoofed, the MAC address of the nmap station will not be spoofed. Close examination of a network trace file on the nmap subnet will clearly show the real hardware address of the spoofed IP addresses.

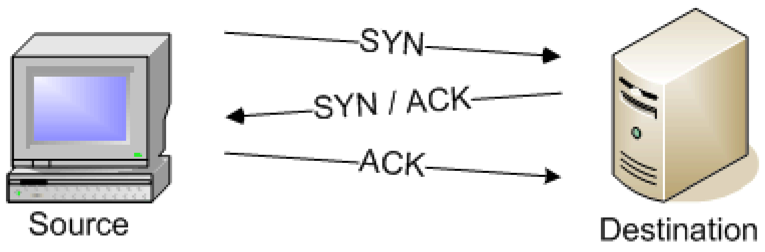
The decoy option is a good example of how "active filtering" can become a detriment on a production network. If decoy stations are used to scan a device, an active firewall reconfiguration or active IPS blocking may prevent legitimate traffic from traversing the network. An overly aggressive active filtering profile can inadvertently create self-inflicted denial of service attacks! This nmap scan can assist network teams with testing and tuning of existing systems to help prevent these situations from occurring.

The Danger of Decoy-Initiated SYN Floods

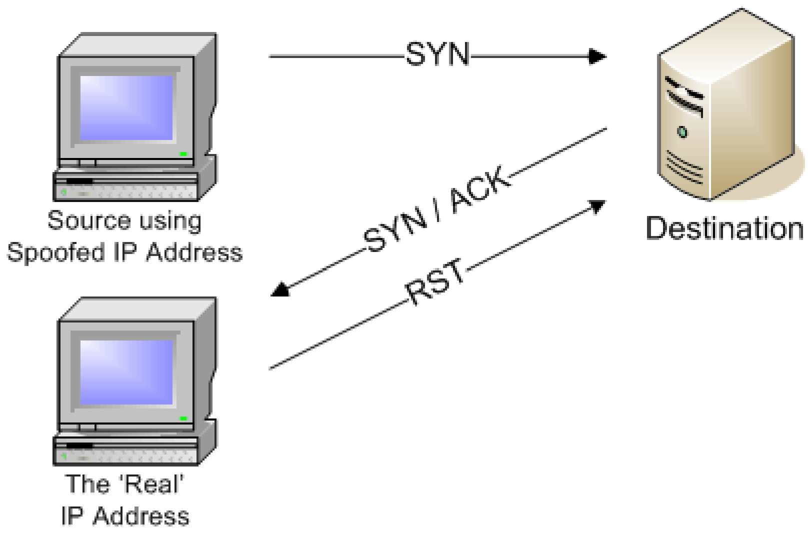
The decoy option (`-D`) should be used with caution! As mentioned on the nmap man page, it's possible to inadvertently SYN flood a destination station if a decoy IP address isn't actually on the network. This could disable a remote device and may require a restart of the remote device before it would be accessible again. The results of a SYN flood are usually unpredictable because of the differences in operating systems and TCP/IP stacks.

A SYN flood is created when a device has received an excessive number of SYN requests. This overwhelming number of SYN packets prevents the station from creating new TCP-based application connections.

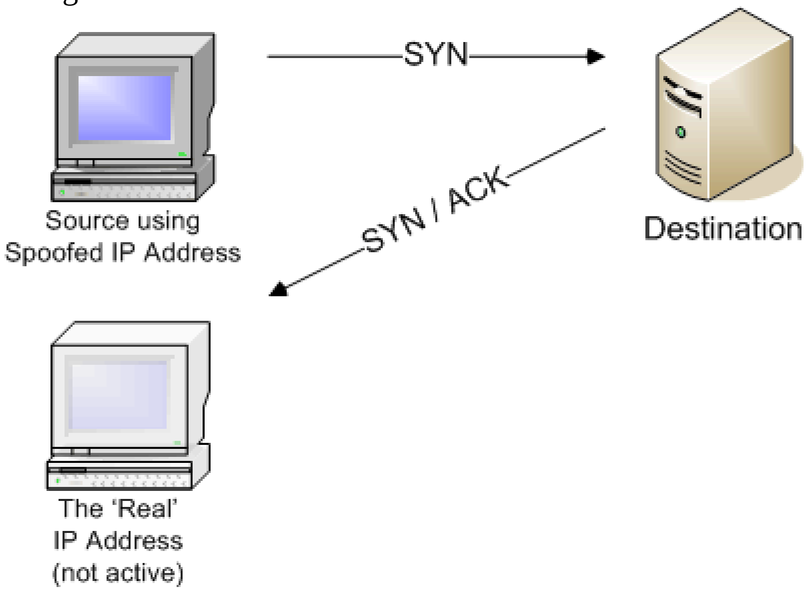
During a normal TCP handshake, this transfer of packets occurs:



A TCP handshake using a spoofed address doesn't operate this way. A connection using a spoofed IP address never completes the TCP handshake. Although the first two frames of the "spoofed" handshake are identical, the second frame is correctly sent to the IP address of the "real" non-spoofed station. Because the real IP address receives an acknowledgement to a SYN that it never sent, it replies with a TCP reset frame. The destination station receives the RST, and releases the resources associated with this halfway-built TCP connection.



If the nmap source spoofs an IP address that isn't on the network, this half-built TCP connection never resets! The SYN/ACK replies are sent to an IP address that isn't on the network, so a RST is never sent back to the destination station. Nmap will continue to spoof packets to the destination station, and the resources available on the destination station will continue to decrease as it waits to complete the TCP handshake. If the destination station's resources are depleted, it will not create new TCP sessions for any nmap-related or legitimate network traffic.


SYN
SYN / ACK
Source using Spoofed IP Address
Destination
The 'Real' IP Address (not active)

This is one of the few instances where an untrained nmap user can really cause some havok on the network! Be careful when using decoys!

Source Address (`-S <IP_address>`)
There may be instances where nmap cannot determine the local IP address of the station where it's running. When this happens, nmap will stop the scan and provide this message:

```
Unable to find appropriate source address and device
interface to use when sending packets to XXX.XXX.XXX.XXX
```

If the local IP address is added to the nmap command line with the `–S` option, this message will not be displayed and the nmap scan will run normally.

As with the decoy option (`-D`), the source address option (`-S`) can be used to spoof the IP address of the nmap station. This can cause SYN flood problems for the remote device, so this spoofing process must be managed very carefully. If the local IP address will be spoofed, the interface option (`-e`) may be required, especially if the spoofed address belongs to a network to which the nmap station is not attached.

Interface (`-e <interface>`)

The interface option is used to force the nmap scan to use a specific hardware interface on the nmap station. If nmap is having problems determining which interface to use, it will provide this error message before quitting:

```
Unable to determine what interface to route packets through
to XXX.XXX.XXX.XXX
QUITTING!
```

In POSIX-based systems, the interface name is the device name that can be seen using the `ifconfig` command. In Windows-based systems, the interface names aren't as standardized. Fortunately, nmap includes a method for Windows-based systems to retrieve the winpcap device name. The following command will list the available Windows interfaces:

```
> nmap --win_list_interfaces
Available interfaces:

Name        Raw mode  IP
loopback0   SOCK_RAW  127.0.0.1
eth0        winpcap   192.168.0.5

>
```

CHAPTER 6:

LOGGING OPTIONS

Nmap provides numerous methods of logging the scan output. Nmap's output logs not only save important scan information, they also provide a method of restarting a previously cancelled scan.

In all of the output formats, the filename can be replaced with a hypen (`-`) to redirect the output to `stdout`. This can be useful if the output information will be used by another piped process on the same command line.

Normal Format (`-oN <logfilename>`)

Nmap's normal output format (`-oN`) saves a similar view of the output that's displayed on the screen during an nmap scan. This readable text is useful when the output is used in a document or printed for later reference.

This saved information is very similar to the run-time output, except for some additional initialization information at the top of the scan:

```
# nmap 3.81 scan initiated Fri May 20 21:00:50 2005 as: ./nmap -v -sS -
oA oA9 192.168.0.9
Interesting ports on 192.168.0.9:
(The 1654 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
79/tcp   open  finger
110/tcp  open  pop3
111/tcp  open  rpcbind
514/tcp  open  shell
886/tcp  open  unknown
2049/tcp open  nfs
MAC Address: 00:03:47:6D:28:D7 (Intel)
```

```
# Nmap run completed at Fri May 20 21:01:01 2005 -- 1 IP address (1
host up)
  scanned in 10.950 seconds
```

This output type doesn't follow any particular format. All of the nmap information is jumbled together onto separate lines of output, with a text layout that differs slightly for every scan. This variability makes it difficult for other tools to process the information contained in the output file.

Most of the examples in this book are taken from a normal format output file!

XML Format (`-oX <logfilename>`)

Extensible Markup Language (XML) is a standard method of describing information, and should not be confused with HyperText Markup Language (HTML). HTML focuses on how data is to be displayed, while XML focuses on describing the data. XML is not a language, XML is a way to structure, store, and send information.

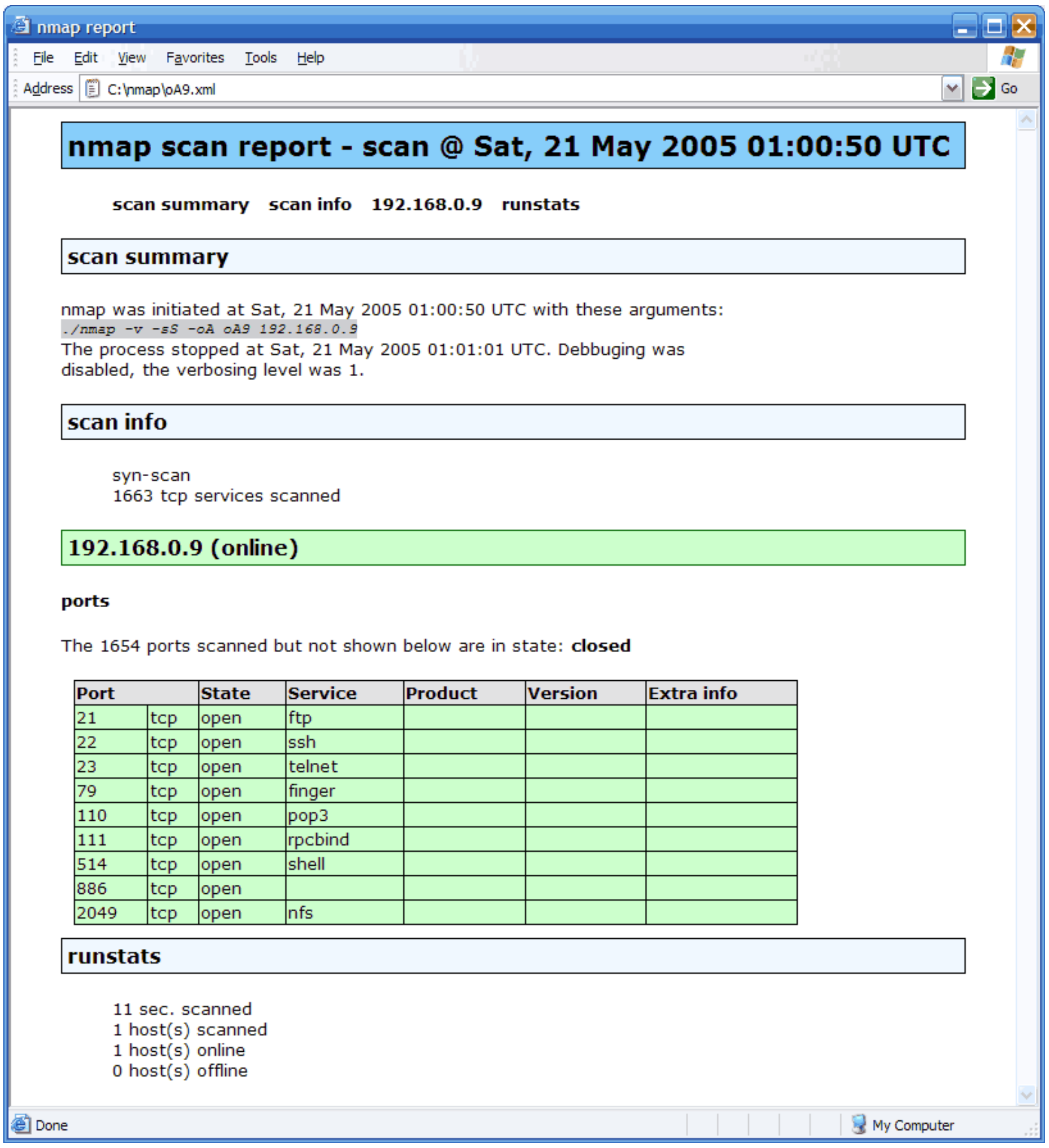
XML is a great format to use when additional processing of nmap data is required. The XML format clearly identifies all of the nmap data, creating an output format that can be parsed and understood without any misinterpretations or inconsistencies. The nmap XML Document Type Definition (DTD) information is contained in a separate file available on the nmap web site at http://www.insecure.org/nmap/data/nmap.dtd.

The XML format contains a standard structure of information, but this standard structure is best read by machines. Nmap's man page refers to the XML output format as the "recommended" format when other programs need to interact with nmap's output. Most humans won't be able to easily obtain information from this XML file:

```
<?xml version="1.0" ?>
<?xml-stylesheet href="http://www.insecure.org/nmap/data/nmap.xsl"
type="text/xsl"?>
<!-- nmap 3.81 scan initiated Fri May 20 21:00:50 2005 as: ./nmap -v -
sS -oA
   oA9 192.168.0.9 -->
<nmaprun scanner="nmap" args="./nmap -v -sS -oA oA9 192.168.0.9"
start="1116637250"
   startstr="Fri May 20 21:00:50 2005" version="3.81"
xmloutputversion="1.01">
<scaninfo type="syn" protocol="tcp" numservices="1663" services="1-
1027,1029-1033,
   1040,1050,1058-1059,1067-1068,1076,1080,1083-1084,1103,1109-
1110,1112,1127,1139,
   1155,1178,1212,1214,1220,1222,1234,1241,1248,1337,1346-1381,1383-
1552,1600,1650-1652,
   1661-1672,1680,1720,1723,1755,1761-1764,1827,1900,1935,1984,1986-
2028,2030,2032-2035,
   2038,2040-2049,2053,2064-2065,2067-2068,2105-2106,2108,2111-
2112,2120-2121,2201,2232,
   2241,2301,2307,2401,2430-2433,2500-2501,2564,2600-2605,2627-
2628,2638,2766,2784,2809,
   2903,2998,3000-3001,3005-
3006,3049,3052,3064,3086,3128,3141,3264,3268-3269,3292,3306,
   3333,3372,3389,3421,3455-3457,3462,3531,3632,3689,3900,3984-
3986,3999-4000,4008,4045,
   4132-
4133,4144,4224,4321,4333,4343,4444,4480,4500,4557,4559,4660,4672,4899,4
987,4998,
   5000-5003,5010-5011,5050,5100-5102,5145,5190-5193,5232,5236,5300-
5305,5308,5400,5405,
   5432,5490,5510,5520,5530,5540,5550,5555,5631-5632,5680,5713-
5717,5800-5803,5900-5903,
   5977-5979,5997-6009,6017,6050,6101,6103,6105-6106,6110-6112,6141-
6148,6346,6400-6401,
   6502,6543-6544,6547-6548,6558,6588,6666-6668,6699,6969,7000-
7010,7070,7100,7200-7201,
   7273,7326,7464,7597,8000,8007,8009,8080-
8082,8443,8888,8892,9090,9100,9111,9152,9535,
   9876,9991-9992,9999-10000,10005,10082-10083,11371,12000,12345-
12346,13701-13702,
   13705-13706,13708-13718,13720-13722,13782-
13783,15126,16959,17007,17300,18000,
   18181-
18185,18187,19150,20005,22273,22289,22305,22321,22370,26208,27000-
27010,27374,
   27665,31337,32770-32780,32786-32787,38037,38292,43188,44334,44442-
44443,47557,49400,
   54320,61439-61441,65301" />
<verbose level="1" />
<debugging level="0" />
<host><status state="up" />
<address addr="192.168.0.9" addrtype="ipv4" />
<address addr="00:03:47:6D:28:D7" addrtype="mac" vendor="Intel" />
<hostnames />
<ports><extraports state="closed" count="1654" />
<port protocol="tcp" portid="21"><state state="open" /><service
name="ftp"
   method="table" conf="3" /></port>
<port protocol="tcp" portid="22"><state state="open" /><service
name="ssh"
   method="table" conf="3" /></port>
<port protocol="tcp" portid="23"><state state="open" /><service
name="telnet"
   method="table" conf="3" /></port>
<port protocol="tcp" portid="79"><state state="open" /><service
name="finger"
   method="table" conf="3" ></port>
<port protocol="tcp" portid="110"><state state="open" /><service
name="pop3"
   method="table" conf="3" /></port>
<port protocol="tcp" portid="111"><state state="open" /><service
name="rpcbind"
   method="table" conf="3" /></port>
<port protocol="tcp" portid="514"><state state="open" /><service
name="shell"
   method="table" conf="3" /></port>
<port protocol="tcp" portid="886"><state state="open" /></port>
<port protocol="tcp" portid="2049"><state state="open" /><service
name="nfs"
   method="table" conf="3" /></port>
</ports>
</host>
<runstats><finished time="1116637261" timestr="Fri May 20 21:01:01
2005"/><hosts
   up="1" down="0" total="1" />
```

```
<!-- Nmap run completed at Fri May 20 21:01:01 2005; 1 IP address (1
host up) scanned in
   10.950 seconds -->
</runstats></nmaprun>
```

Fortunately for humans, nmap includes an Extensible Stylesheet Language (XSL) file that assists in translating the XML information into a viewable HTML format. In most cases, the XML file can be opened in any browser to display the translated format:



nmap scan report - scan @ Sat, 21 May 2005 01:00:50 UTC

scan summary **scan info** **192.168.0.9** **runstats**

scan summary

nmap was initiated at Sat, 21 May 2005 01:00:50 UTC with these arguments:
`./nmap -v -sS -oA oA9 192.168.0.9`
The process stopped at Sat, 21 May 2005 01:01:01 UTC. Debbuging was disabled, the verbosing level was 1.

scan info

syn-scan
1663 tcp services scanned

192.168.0.9 (online)

ports

The 1654 ports scanned but not shown below are in state: **closed**

Port		State	Service	Product	Version	Extra info
21	tcp	open	ftp			
22	tcp	open	ssh			
23	tcp	open	telnet			
79	tcp	open	finger			
110	tcp	open	pop3			
111	tcp	open	rpcbind			
514	tcp	open	shell			
886	tcp	open				
2049	tcp	open	nfs			

runstats

11 sec. scanned
1 host(s) scanned
1 host(s) online
0 host(s) offline

After viewing the XML/XSL file output, it should be clear that this format has some obvious advantages. Security managers who have had to previously sift through pages of plain text can appreciate the more practical formatting and clearer output functionality available with the XML output.

Does your nmap output look this good?

Stylesheet (`--stylesheet <filename>`)

When using the XML output format (`-oX`), nmap automatically includes a reference to the `nmap.xsl` file that was included with nmap. Viewing the XML output file in most browsers will automatically load the stylesheet and translate the XML information into HTML format.

There may be cases where another stylesheet would be used to translate the XML information. For example, a security manager may want to change the header of the page to include company information, add logos, or change the colors of the HTML output. If the XML information will be viewed in a report format, this capability can be very beneficial.

The nmap web site always maintains the latest version of the default stylesheet. The `--stylesheet` option can reference the XSL file at http://www.insecure.org/nmap/data/nmap.xsl, or the option can point to an XSL file on a local device or web server.

No Stylesheet (`--no-stylesheet`)

There may be occasions where the XML file will be used by another application and not require a conversion to HTML. In these cases, it's more efficient to create the XML file without any references to a stylesheet. This option is only applicable when the XML output format (`-oX`) is chosen, although nmap will still run without an error if the `--no-stylesheet` option is used without any reference to XML output.

Unlike other multi-word nmap options, the `--no-stylesheet` option separates the words with a hypen (`-`) instead of an underscore (_). Nmap will not recognize this option if an underscore is used.

Grepable Format (`-oG <logfilename>`)

The unix command "grep" is well known for its ability to quickly locate text within a file. The "grep" moniker is an abbreviation for the name "Global Regular Expression Print."

The "grepable" nmap format places the scan summary information and the host scan results onto single lines. This makes it very easy to search for specific information, since the results of a text search would identify a single applicable line of text. This also makes it relatively easy for other programs to query and interact with nmap's output.

An excerpt of a scan's grepable output is listed here, although some of the lines are truncated to fit onto the screen. The nmap operational output lines begin with pound signs (#), and the host scan information line begins with the word "`Host:`"

```
# nmap 3.81 scan initiated Sat May 21 09:42:40 2005 as: ./nmap -v -oA
subnet0 192.168.0.1/24
# Ports scanned: TCP(1663;1-1027,1029-1033,1040,1050,1058-1059,1067-
1068,1076,1080,1083-1084...
```

```
Host: 192.168.0.0 () Ports: 80/open/tcp//http///  Ignored State: closed
(1662)
Host: 192.168.0.1 () Ports: 80/open/tcp//http///  Ignored State: closed
(1662)
Host: 192.168.0.3 () Ports: 80/open/tcp//http///,
443/open/tcp//https/// Ignored State: filtered (1661)
Host: 192.168.0.6 () Ports: 80/open/tcp//http///,
443/open/tcp//https/// Ignored State: filtered (1661)
Host: 192.168.0.7 () Ports: 68/open/tcp//dhcpclient///,
6000/open/tcp//X11/// Ignored State: closed (1661)
Host: 192.168.0.99 ()  Ports: 21/open/tcp//ftp///,
23/open/tcp//telnet///, 25/filtered/tcp//smtp///...
Host: 192.168.0.255 () Ports: 80/open/tcp//http///    Ignored State:
closed (1662)
# Nmap run completed at Sat May 21 09:43:40 2005--256 IP addresses (7
hosts up) scanned in 59.915 seconds
```

Some scan information will not be available in the grepable output file, and the nmap man page makes it clear that the recommended output format for after-scan interaction is the XML output format (`-oX`).

Although the XML format is the recommended output format, features such as the `--resume` option will only operate correctly with the grepable format (`-oG`) or the normal format (`-oN`).

The grepable output type was previously called the "machine readable" format (`-oM`). The `-oM` option will still work, but the alias has been deprecated and shouldn't be used.

The grepable format is always variable, and there's no guaranteed method of automatically retrieving information from a grepable text file. Simple changes to the scan, such as adding the verbose option (`-v`), will add variability to the output. If the output will undergo extensive post-scan analysis, a better option is to use the XML output (`-oX`) option.

All Formats (`-oA <basefilename>`)

If all of the formats are required, the "all formats" option (`-oA`) will create three separate output files. The basefilename will be added to files that have an extension of .nmap (normal output), .gnmap (grepable output), and .xml (XML output).

This format is helpful when both the XML output is needed and the `--resume` option is required. Since `--resume` only works with normal or grepable output, this "all formats" option covers all circumstances.

Get used to using the "all formats" output option for all of your scans!

Script Kiddie Format (`-oS <logfilename>`)

The "script kiddie" format (`-oS`) has very little practical functionality. Fyodor added this output format to nmap as a joke, since most users of "leet speak" (or l33t 5p34k) are considered to be technically immature. Fyodor considers the use of nmap to be for serious security purposes, and his sarcastic dig at the less technical script kiddies is intended to be as pointed as it is accurate.

There actually is some practical use to this format, since it takes advantage of a flexible output system that could be used in the future for other purposes. Fyodor's opinion of this format can be found at this nmap-hackers mailing list archive page:

http://seclists.org/lists/nmap-hackers/2000/Jan-Mar/0005.html

---

HTML Format (`-oH`)
The HTML format command line is included in the nmap source, but it does not include any supporting code. This format isn't currently supported, and if the option is used in the command line this message will display:

```
HTML output is not yet supported
QUITTING!
```

Considering that XML format has now been integrated into nmap with cooresponding XSL stylesheets, a separate HTML format will probably be unnecessary.

Resume Scan (`--resume <logfilename>`)
A scan can be interrupted during processing with the control-C key combination. The `--resume` option allows the nmap scan to restart without reprocessing previously scanned devices.

The resume scan option works by parsing a normal (`-oN`) or grepable (`-oG`) output file to determine the last successful scanned device. Since the original nmap options are also included in the log file, no additional nmap parameters are required or allowed on the command line when resuming a scan.

Once the previously saved output file has been parsed, the `--resume` option can determine the last successfully scanned device and continue the scanning process from that point. If a device was partially scanned prior to the interruption, it will be scanned again from the very beginning of the process.

Ironically, the recommended output format for post-processing of nmap scans is the XML output (`-oX`), but the `--resume` option will not work with an XML file. If both XML output and the ability to resume a scan is required, consider using the all formats (`-oA`) output option.

If a scan is interrupted that used the `--randomize_hosts` option, nmap has no method to recreate the same randomness that was used in the initial scan. If this type of scan is resumed, nmap will provide this message:

```
WARNING: You are attempting to resume a scan which used
--randomize_hosts. Some hosts in the last randomized batch
make (sic) be missed and others may be repeated once
```

Append Output (`--append_output`)
If a single log file is required from multiple nmap scans, the `--append_output` option will append the output the to the log filename used on the nmap command line. If this option isn't specified, nmap will overwrite an existing logfile without any warning or message.

CHAPTER 7:

REAL-TIME INFORMATION OPTIONS
During an nmap scan, there's a lot of activity happening under the hood. These real-time information options provide the nmap user with customizable output relating to the scan and its processing. Some of these options should be used with every scan, and some of these options will never be utilized by most nmap users.

Verbose Mode (`--verbose, -v`)
Nmap's verbose mode provides the option of obtaining additional detail in the scan output. The verbose mode doesn't change what occurs during the scan, it only changes the amount of information that nmap displays on its output.

There are three level of verbosity; none, level 1, and level 2. Level 1 verbosity displays nearly all verbose text, and level 2 includes TCP IPID sequencing information, version scanning service matching limits, and additional OS fingerprinting details. In most cases, level two verbosity can be selected for every nmap scan without creating an overwhelming amount of output.

The verbosity levels are based on how many times the `--verbose` option is invoked on the nmap command line. Although it's possible to request a verbosity level higher than 2, there doesn't appear to be anything in the source code that would support any additional verbosity.

The following nmap command lines all specify level 2 verbosity:
```
# nmap 192.168.0.1 --verbose --verbose
# nmap 192.168.0.1 -v -v
# nmap 192.168.0.1 -v -verbose
# nmap 192.168.0.1 -vv
```
When the XML output option is selected (`-oX`), the verbosity level is written to the XML file. The value is referenced in the "verbose level" tag:
```
<verbose level="1" />
```

A single use of the `--debug` option will increase the verbosity level by one. Conversely, increasing the verbosity level does not affect the debug level.

The `--verbose` option doesn't add a lot of text to nmap's `stdout` output, and it's sometimes useful to have the additional information after the nmap scan is complete. Since the verbose information isn't added to the nmap output files (i.e, `-oA`), the nmap `stdout` output should be piped to a file from the nmap command line. In most cases, the `-vv` option should be used unless detailed nmap statistics aren't required.

Version Trace (`--version_trace`)

Using the `--version_trace` option during a version detection scan (`-sV`) creates an extensive trace that details the versioning processes through every step. As nmap queries the application on the remote device, the application requests and responses are written to `stdout`.

This information is never included in an nmap output file (i.e., `-oA`). If the version trace information needs to be referenced later, the nmap stdout output should be piped to a file from the nmap command line. One simple method to do this is to use a greater-than sign to redirect the output:

```
# nmap –vv –sV 192.168.0.1 –oA output --version_trace > nmap_output.txt
```

An excerpt of the `--version_trace` output is shown below:

```
Initiating service scan against 1 service on 192.168.0.1 at 17:10
NSOCK (1.5520s) TCP connection requested to 192.168.0.1:80 (IOD #1) EID
8
NSOCK (1.5520s) nsock_loop() started (no timeout). 1 events pending
NSOCK (1.5550s) Callback: CONNECT SUCCESS for EID 8 [192.168.0.1:80]
NSOCK (1.5550s) Read request from IOD #1 [192.168.0.1:80] (timeout:
5000ms) EID 18
NSOCK (6.5540s) Callback: READ TIMEOUT for EID 18 [192.168.0.1:80]
NSOCK (6.5540s) Write request for 18 bytes to IOD #1 EID 27
[192.168.0.1:80]: GET / HTTP/1.0....
NSOCK (6.5540s) Read request from IOD #1 [192.168.0.1:80] (timeout:
5000ms) EID 34
NSOCK (6.5550s) Callback: WRITE SUCCESS for EID 27 [192.168.0.1:80]
NSOCK (6.6430s) Callback: READ SUCCESS for EID 34 [192.168.0.1:80] (123
bytes)
NSOCK (6.6430s) Read request from IOD #1 [192.168.0.1:80] (timeout:
4910ms) EID 42
NSOCK (6.6670s) Callback: READ SUCCESS for EID 42 [192.168.0.1:80] (16
bytes): 401 Unauthorized
NSOCK (6.6670s) Read request from IOD #1 [192.168.0.1:80] (timeout:
4886ms) EID 50
NSOCK (6.6690s) Callback: READ EOF for EID 50 [192.168.0.1:80]
NSOCK (6.6690s) TCP connection requested to 192.168.0.1:80 (IOD #2) EID
56
NSOCK (6.6700s) Callback: CONNECT SUCCESS for EID 56 [192.168.0.1:80]
NSOCK (6.6700s) Write request for 22 bytes to IOD #2 EID 67
[192.168.0.1:80]: OPTIONS / HTTP/1.0....
NSOCK (6.6700s) Read request from IOD #2 [192.168.0.1:80] (timeout:
5000ms) EID 74
NSOCK (6.6710s) Callback: WRITE SUCCESS for EID 67 [192.168.0.1:80]
NSOCK (6.7400s) Callback: READ EOF for EID 74 [192.168.0.1:80]
NSOCK (6.7400s) TCP connection requested to 192.168.0.1:80 (IOD #3) EID
80
NSOCK (6.7420s) Callback: CONNECT SUCCESS for EID 80 [192.168.0.1:80]
```

Although the version trace information shows similar information to the `--packet_trace` function, this version trace output does not display a packet-level view of the application conversation. Even with `--packet_trace` enabled, the version detection output will never include packet-level detail in neither nmap output files nor `stdout` output.

Packet Trace (--packet_trace)

The `--packet_trace` option provides a network-level packet display of nmap's conversations during the scan process. This includes a timestamp, IP addresses, port numbers, and protocol header information. This option is extremely useful for learning more about how nmap interacts with remote devices. This can be used as a powerful educational tool for any network professional!

The packet trace option shows network interactions only, not nmap's internal debugging information. However, if the debug option level is greater than 2, the `--packet_trace` option will be enabled automatically.

The `--packet_trace` output is displayed to `stdout` and can also be saved to nmap's normal (`-oN`) output file. This packet trace information will not be saved into greppable (`-oG`) or XML (`-oX`) output files.

An excerpt from the standard output shows the extensive network information contained in the packet trace:

```
SENT (0.0340s) ICMP 192.168.0.7 > 192.168.0.1 Echo request
(type=8/code=0) ttl=57 id=55712 iplen=28
SENT (0.0350s) TCP 192.168.0.7:57019 > 192.168.0.1:80 A ttl=53 id=54387
iplen=40 seq=3243190494 win=2048 ack=2504992990
RCVD (0.0350s) ICMP 192.168.0.1 > 192.168.0.7 Echo reply
(type=0/code=0) ttl=64 id=16746 iplen=28
SENT (0.5710s) TCP 192.168.0.7:56996 > 192.168.0.1:3389 S ttl=44
id=14391 iplen=40 seq=860312039 win=1024
SENT (0.5710s) TCP 192.168.0.7:56996 > 192.168.0.1:80 S ttl=55 id=29561
iplen=40 seq=860312039 win=4096
SENT (0.5710s) TCP 192.168.0.7:56996 > 192.168.0.1:636 S ttl=51
id=60961 iplen=40 seq=860312039 win=4096
SENT (0.5710s) TCP 192.168.0.7:56996 > 192.168.0.1:23 S ttl=44 id=26608
iplen=40 seq=860312039 win=1024
SENT (0.5710s) TCP 192.168.0.7:56996 > 192.168.0.1:21 S ttl=46 id=52419
iplen=40 seq=860312039 win=3072
SENT (0.5710s) TCP 192.168.0.7:56996 > 192.168.0.1:113 S ttl=48
id=42709 iplen=40 seq=860312039 win=1024
SENT (0.5710s) TCP 192.168.0.7:56996 > 192.168.0.1:389 S ttl=54
id=37581 iplen=40 seq=860312039 win=3072
SENT (0.5710s) TCP 192.168.0.7:56996 > 192.168.0.1:256 S ttl=58
id=56038 iplen=40 seq=860312039 win=3072
SENT (0.5710s) TCP 192.168.0.7:56996 > 192.168.0.1:53 S ttl=40 id=9325
iplen=40 seq=860312039 win=1024
SENT (0.5720s) TCP 192.168.0.7:56996 > 192.168.0.1:1723 S ttl=42
id=64956 iplen=40 seq=860312039 win=3072
```

```
RCVD (0.5720s) TCP 192.168.0.1:3389 > 192.168.0.7:56996 RA ttl=64
id=16750 iplen=40 seq=0 win=0 ack=860312039
RCVD (0.5720s) TCP 192.168.0.1:80 > 192.168.0.7:56996 SA ttl=64
id=16751 iplen=44 seq=28372096 win=3072 ack=860312040
RCVD (0.5730s) TCP 192.168.0.1:636 > 192.168.0.7:56996 RA ttl=64
id=16752 iplen=40 seq=0 win=0 ack=860312039
RCVD (0.5730s) TCP 192.168.0.1:23 > 192.168.0.7:56996 RA ttl=64
id=16753 iplen=40 seq=0 win=0 ack=860312039
RCVD (0.5740s) TCP 192.168.0.1:21 > 192.168.0.7:56996 RA ttl=64
id=16754 iplen=40 seq=0 win=0 ack=860312039
RCVD (0.5750s) TCP 192.168.0.1:113 > 192.168.0.7:56996 RA ttl=64
id=16755 iplen=40 seq=0 win=0 ack=860312039
RCVD (0.5750s) TCP 192.168.0.1:389 > 192.168.0.7:56996 RA ttl=64
id=16756 iplen=40 seq=0 win=0 ack=860312039
RCVD (0.5750s) TCP 192.168.0.1:256 > 192.168.0.7:56996 RA ttl=64
id=16757 iplen=40 seq=0 win=0 ack=860312039
RCVD (0.5760s) TCP 192.168.0.1:53 > 192.168.0.7:56996 RA ttl=64
id=16758 iplen=40 seq=0 win=0 ack=860312039
RCVD (0.5770s) TCP 192.168.0.1:1723 > 192.168.0.7:56996 RA ttl=64
id=16759 iplen=40 seq=0 win=0 ack=860312039
SENT (0.5770s) TCP 192.168.0.7:56996 > 192.168.0.1:22 S ttl=48 id=11606
iplen=40 seq=860312039 win=1024
SENT (0.5770s) TCP 192.168.0.7:56996 > 192.168.0.1:554 S ttl=46
id=53764 iplen=40 seq=860312039 win=3072
SENT (0.5770s) TCP 192.168.0.7:56996 > 192.168.0.1:443 S ttl=59
id=34121 iplen=40 seq=860312039 win=4096
SENT (0.5770s) TCP 192.168.0.7:56996 > 192.168.0.1:25 S ttl=49 id=45437
iplen=40 seq=860312039 win=2048
SENT (0.5770s) TCP 192.168.0.7:56996 > 192.168.0.1:428 S ttl=52
id=19724 iplen=40 seq=860312039 win=1024
SENT (0.5770s) TCP 192.168.0.7:56996 > 192.168.0.1:2 S ttl=38 id=27980
iplen=40 seq=860312039 win=3072
SENT (0.5770s) TCP 192.168.0.7:56996 > 192.168.0.1:818 S ttl=50
id=57246 iplen=40 seq=860312039 win=3072
SENT (0.5770s) TCP 192.168.0.7:56996 > 192.168.0.1:1492 S ttl=45
id=19237 iplen=40 seq=860312039 win=2048
SENT (0.5780s) TCP 192.168.0.7:56996 > 192.168.0.1:958 S ttl=54
id=60894 iplen=40 seq=860312039 win=3072
SENT (0.5780s) TCP 192.168.0.7:56996 > 192.168.0.1:754 S ttl=49
id=46608 iplen=40 seq=860312039 win=2048
SENT (0.5780s) TCP 192.168.0.7:56996 > 192.168.0.1:3531 S ttl=45
id=8839 iplen=40 seq=860312039 win=2048
SENT (0.5780s) TCP 192.168.0.7:56996 > 192.168.0.1:720 S ttl=38
id=31256 iplen=40 seq=860312039 win=3072
```

If a version scan (`-sV`) is combined with the `--packet_trace` option, the `--version_trace` option is automatically enabled.

Debug Mode (`--debug, -d`)

Nmap's debug mode displays extensive internal nmap information, and the amount of detail depends on the debug level selected on the nmap command line. There are five debug levels ranging from zero through four. Although it's possible to request a debug level higher than 4, there doesn't appear to be anything in the source code to support additional debugging output.

The `--debug` output is displayed on `stdout`, but will not be saved into any of nmap's output files (i.e., `-oA`). If the debug information is important, the nmap `stdout` output should be piped to a file from the nmap command line.

Increasing the debug levels will increase the amount of information in the nmap output. For example, if the debug option level is greater than two, the `--packet_trace` option will be enabled automatically.

The `--debug` syntax is different than what's used for the verbose (`--verbose`) option. The following nmap command lines all specify a debug level of two:

```
# nmap 192.168.0.1 --debug --debug
# nmap 192.168.0.1 -d -d
# nmap 192.168.0.1 -d --debug
```

Unlike the `--verbose` option, the syntax of `-dd` is not valid and will not register any debug levels during the nmap scan. This will not give an error, however, and the nmap scan will continue to run normally without providing any debug output.

Increasing the debug level by one will also increase the verbosity level by one. If a verbosity level has already been set, it will be incremented again if the `--debug` option is also included on the nmap command line. For example, the following nmap command line runs with a verbosity level of three and a debugging level of two:

```
# nmap 192.168.0.1 –d –debug –v
```

This output shows an nmap SYN scan without any debug level:

```
# ./nmap -sS -oA debug_sS_off 192.168.0.1

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-05-25
23:26 EDT
Interesting ports on 192.168.0.1:
(The 1662 ports scanned but not shown below are in state: closed)
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:09:5B:D4:BB:FE (Netgear)

Nmap finished: 1 IP address (1 host up) scanned in 1.988 seconds
#
```

If the debug command is enabled, additional information is displayed. This output shows a debug level one output from the same SYN scan command:

```
# ./nmap -sS --debug -oA debug_sS 192.168.0.1

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-05-25
23:25 EDT
Packet capture filter (device eth0): (icmp and dst host 192.168.0.7) or
((tcp or udp) and dst host 192.168.0.7 and ( dst port 45024 or dst port
45025 or dst port 45026 or dst port 45027 or dst port 45028))
We got a ping packet back from 192.168.0.1: id = 45708 seq = 33180
checksum = 52182
Hostupdate called for machine 192.168.0.1 state UNKNOWN/COMBO ->
HOST_UP (trynum 0, dotimeadj: yes time: 821)
Finished block: srtt: 778 rttvar: 5000 timeout: 100000 block_tries: 1
up_this_block: 1 down_this_block: 0 group_sz: 1
massping done:  num_hosts: 1  num_responses: 1
Initiating SYN Stealth Scan against 192.168.0.1 [1663 ports] at 23:25
Packet capture filter (device eth0): dst host 192.168.0.7 and (icmp or
(tcp and (src host 192.168.0.1)))
Discovered open port 80/tcp on 192.168.0.1
The SYN Stealth Scan took 1.07s to scan 1663 total ports.
Host 192.168.0.1 appears to be up ... good.
Interesting ports on 192.168.0.1:
(The 1662 ports scanned but not shown below are in state: closed)
```

```
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:09:5B:D4:BB:FE (Netgear)
Final times for host: srtt: 21760 rttvar: 4482  to: 100000

Nmap finished: 1 IP address (1 host up) scanned in 1.867 seconds
               Raw packets sent: 1665 (66.6KB) | Rcvd: 1664 (76.5KB)
#
```

As this output shows, extensive internal nmap operations are detailed during the scan. These internal processes are useful for development purposes, but the output is rarely useful for normal host and network scanning operations.

Interactive Mode (`--interactive`)

Nmap's interactive mode isn't mentioned in the nmap man page, but it is identified on the online quick reference screen (`-h`). The interactive mode is included with nmap to assist with managing an nmap session and to provide an easier method of spoofing the nmap session to be more invisible on a system.

The commands available in `--interactive` mode are similar, but not identical, to the commands available from the nmap command line. For example, the `--spoof` command in interactive mode is comparable to the quash argument vector (`-q`) command on the nmap command line.

This interactive mode can also be called automatically if the nmap binary is renamed to one of many different options. These name options are BitchX, Calendar, X, awk, bash, bash2, calendar, cat, csh, elm, emacs, ftp, fvwm, g++, gcc, gimp, httpd, irc, man, mutt, nc, ncftp, netscape, perl, pine, ping, sleep, slirp, ssh, sshd, startx, tcsh, telnet, telnetd, tia, top, vi, vim, xdvi, xemacs, xterm, and xv. As this list implies, renaming nmap to a less conspicuous name can increase security while simultaneously providing a simple method of starting nmap in interactive mode.

The online interactive help screen is shown below. As the examples at the bottom of the help screen show, the commands and functionality in interactive mode are similar to those on a normal nmap scan:

```
# ./nmap --interactive

Starting nmap V. 3.81 ( http://www.insecure.org/nmap/ )
Welcome to Interactive Mode -- press h <enter> for help
nmap> h
Nmap Interactive Commands:
n <nmap args> -- executes an nmap scan using the arguments given and
waits for nmap to finish.  Results are printed to the
screen (of course you can still use file output commands).
! <command>   -- runs shell command given in the foreground
x             -- Exit Nmap
f [--spoof <fakeargs>] [--nmap_path <path>] <nmap args>
-- Executes nmap in the background (results are NOT
printed to the screen).  You should generally specify a
file for results (with -oX, -oG, or -oN).  If you specify
fakeargs with --spoof, Nmap will try to make those
appear in ps listings.  If you wish to execute a special
version of Nmap, specify --nmap_path.
```

```
n -h          -- Obtain help with Nmap syntax
h             -- Prints this help screen.
Examples:
n -sS -O -v example.com/24
f --spoof "/usr/local/bin/pico -z hello.c" -sS -oN e.log example.com/24

nmap>
```

Noninteractive Mode (`--noninteractive`)

Nmap's non-interactive mode doesn't currently provide any additional functionality, since the `--noninteractive` option is already the default operation. If future nmap functionality requires user input, this option could be used to force nmap into a non-interactive option that would be useful for unattended nmap scripting.

Chapter 8: Tuning and Timing Options

Eventually, a network propeller-head will need to modify the bits and bytes of frames used during an nmap session. Fortunately, nmap provides extensive tweaking and tuning capabilities to perfectly match the desires of the enterprising security manager.

This tutorial has separated these tweaks into two categories; packet tuning, and timing options. Nmap packet tuning relates to the information contained in the network packets, and nmap's timing options relate to the intricate tweaks that affect the speed and delay associated with the scanning process.

Nmap Packet Tuning

Nmap's packet tuning options can customize individual header values, the total number of packet fragments, or the size of the frames used in an nmap scan. If the packets need to be altered, these tweaks will provide many options!

Time to Live (`--ttl <value>`)

The time to live (TTL) value is a one-byte field in the IP header that is used to limit the time and distance a packet can travel through the network. As an IP frame traverses a gateway, the time to live value is decreased by one. Once the TTL reaches zero, it is dropped by the gateway and an ICMP Time Exceeded message is sent to the originating station.

The TTL field provides a valuable service by dropping frames that may be circling the network due to a routing loop or a routing table misconfiguration. Without the TTL process, these packets would circle the network indefinitely!

The TTL value can be any number between 0 and 255. Most stations set their TTL to a relatively large number, such as 32 or 64. There's no standard value, and rarely do workstations need to modify TTL settings.

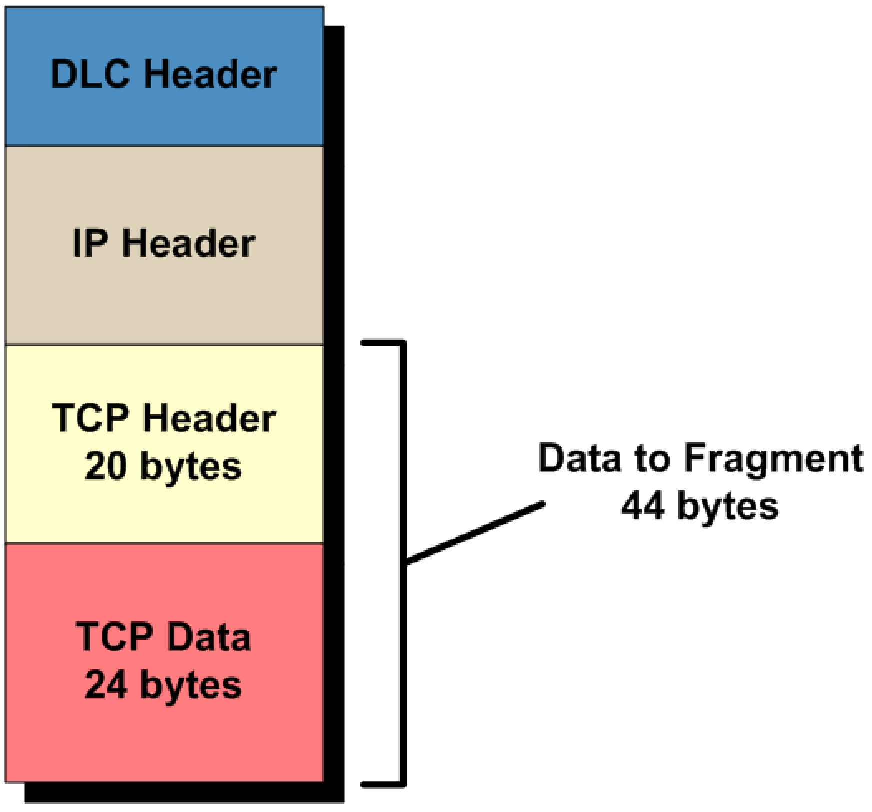
In nmap, the TTL option (`--ttl`) can be used to administratively limit the distance that a packet can travel. If nmap's scanning process should stay on the local network instead of traversing a wide area network link, the TTL can be changed to a value low enough to remain off of the WAN. This low TTL value would cause the IP gateway to drop the packet before it's sent across the slower WAN link.

Using a TTL of zero will ensure that no packets leave the immediate IP subnet!

Use Fragmented IP Packets (`-f, -ff`)
A fragment is an IP frame that has been split into smaller pieces. This process is normally used to traverse links that can't handle large frame sizes, such as WAN links. During normal operation, this process is invisible to the end user.

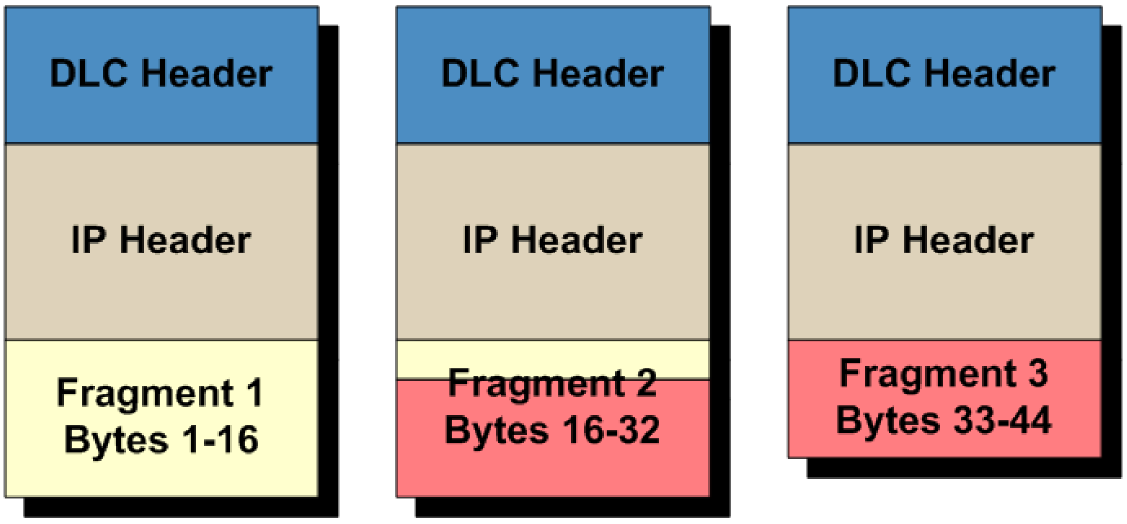
When a station fragments data, each fragmented piece of information has its own IP header. All of the fragmented packets are sent across the network and are not (usually) reconstructed until they all arrive at the remote device. Under most circumstances, the remote device rebuilds the original frame from the fragments, but some networks have firewalls, intrusion detection systems, or proxy servers that will rebuild the frame for security testing before sending the fragments (or the rebuilt packet information) to the remote device.

The IP Fragmentation Process
The fragmentation process occurs at the IP layer, so everything underneath the IP header will be fragmented. In this example, the IP header is transporting 24 bytes of TCP data. Since both the 20 byte TCP header and the 24 bytes of TCP data will be fragmented, the total amount of fragmented data will total 44 bytes:



Due to the number of fragmentation offset bits available in the IP header, the size of the fragments must be a multiple of eight. This example will use a fragment size of 16 bytes.

Once the packet is fragmented, it retains a DLC header and an IP header but the data underneath is now split into smaller pieces. Since the fragment size is 16 bytes and the total amount of data to fragment totals 44 bytes, the fragmentation process will create three frames:



Notice that the TCP header (colored yellow) spans two fragments, but the TCP header isn't rebuilt in each frame. Instead, the TCP header and the TCP data are contiguously added to the subsequent fragments. The last fragment of data consists of only 12 bytes, but the final packet doesn't need to completely fill the 16 byte fragment size. The last 12 bytes are simply attached to the last frame of the fragmentation.

How to Fragment IP Data with nmap
Unless one of nmap's fragmentation options is specified, nmap sends non-fragmented frames to remote devices. Nmap's fragmentation option (`-f`) is a simple method of sending nmap data with the smallest possible fragmentation value of eight bytes.

The `–ff` fragmentation option will double the fragmentation value to the second-smallest value, or 16 bytes. Nmap doesn't have an `–fff` option, so 16 bytes is the maximum fragmentation size available from these options.

If additional fragmentation options are required, the `--mtu` option can be used to create any fragmentation value.

```
# nmap –sS 192.168.0.1 –f
```
is the same as
```
# nmap –sS 192.168.0.1 --mtu 8
```

and

```
#nmap –sS 192.168.0.1 –ff
```

is the same as

```
# nmap –sS 192.168.0.1 --mtu 16
```

Using IP Fragmentation
The usefulness of this option is a bit of a toss-up. In nmap, a major goal of fragmenting data is to obfuscate the IP data so that it can traverse the network without any filtering or alteration. However, many firewalls and security devices reconstruct fragmented packets so that security decisions can be associated with the data.

This option must run as a privileged user, and it isn't available on all operating systems or in all circumstances. If an operating system other than Linux or one of the BSD flavors is used (such as Windows XP), this message is displayed:

```
Warning: Packet fragmentation selected on a host other than Linux,
OpenBSD, Free
BSD, or NetBSD.  This may or may not work.
```

I tested this option on Windows XP Service Pack 2, and the `–f` and `-ff` options successfully fragmented the packets.

Maximum Transmission Unit (`--mtu <databytes>`)
The maximum transmission unit (MTU) option (`--mtu`) is an extension of nmap's fragmentation options (`-f, -ff`). The `--mtu` option allows any valid data size value, not just 8 or 16 bytes. A fragmentation option of `–f` is the same as specifying an `--mtu` of 8, and a fragmentation option of `-ff` is the same as an `--mtu` of 16.

The `--mtu` option must be greater than zero and be specified in multiples of 8. If the MTU specification is larger than the IP data size, then the frame will not be fragmented.

Data Length (`--data_length <databytes>`)
In most cases, nmap only sends frames with the smallest possible size. The IP data length option (`--data_length`) can expand the frames to any user-defined size.

The data length option appends a defined number of random bytes after the TCP or UDP header of the frame. This `<databyte>` variable does not include the IP or TCP/UDP header.

Nmap Timing Options
It may not be the contents of the nmap scan that need to be modified, but the speed at which the scan operates. Sometimes a scan should run slowly and quietly, and other times the scan should run as quickly and loudly as possible. For all of these situations, nmap provides a plethora of timing options that can affect almost every aspect of the scan.

Unlike other nmap commands, the location of these timing options on the command line is very important. If two conflicting timing options are specified on the command line, the last option has the priority. Although this sounds restrictive, it actually provides more functional timing options. For example, a timing policy (`-T`) could be specified first, and a `--scan_delay` option might be included at the end of the same nmap command line.

This unique combination would combine the predefined timing option functions with the "tweak" of an alternative minimum scanning delay.

Most of nmap's timing options are separated into four categories; round trip time, parallel host scanning, parallel port scanning, and delay. Each of these categories can be configured separately, or a predefined nmap timing policy can be used to specify multiple categories simultaneously.

Host Timeout (`--host_timeout <milliseconds>`)
The host timeout option (`--host_timeout`) doesn't fall into one of the four timing categories. The host timeout is the amount of time an nmap scan will wait before "giving up" on an IP address. When nmap begins the scan of a host, it starts a timer. If the timer reaches the `--host_timeout` value, the scan to that host will immediately halt. If additional hosts are included in the nmap scan options, the scan will continue with those devices.

This timeout value could be related to the speed of the link or the willingness of the remote station to respond in a timely manner. Regardless of the reason, the `--host_timeout` option provides the nmap user with a way to "hurry along" the entire scan process when it hits a single slow device.

The default value for `--host_timeout` is zero. This means that nmap will never abort a scan due to time, no matter how long it may be take to complete. The minimum host timeout amount can be 200 milliseconds, and the maximum can be around 4 billion milliseconds – about a month! Hopefully, most nmap scans won't need this long to complete a scan against a single host.

Setting this value to a relatively low number (30,000 milliseconds) could speed up the completion of a scan where many of the remote devices communicate across slow links or perform aggressive throttling of ICMP or network responses. If these slower devices need to be scanned, `--host_timeout` can be tweaked to a larger number until the devices are scanned successfully at the highest possible timeout value. If only the most responsive devices are to be scanned, this timeout value can be decreased in an effort to speed up the overall nmap scanning process.

Round Trip Time
Simply put, round trip time is calculated as the number of milliseconds required to receive a response to an nmap request. Nmap refers to these as "probe responses," and it keeps a running total of their round trip time and variability. As the scan progresses, nmap will dynamically adjust the round trip time to use for comparison with the minimum and maximum values.

Initial Round Trip Time Timeout (`--initial_rtt_timeout`

`<milliseconds>`)
The `--initial_rtt_timeout` is the starting point for an nmap scan. Nmap can usually determine an accurate round trip time during the nmap ping. If nmap doesn't ping the remote device (option `–P0`), the `--initial_rtt_timeout` option can provide nmap with a timing foundation on which the rest of the scan will build.

The default value for the `--initial_rtt_timeout` is 1,000ms (one second). If this value will be configured manually, it must be greater than zero.

Since this value changes throughout the scan, nmap only uses this option as a starting point. A starting value that is relatively close to the actual response time will increase the efficiency of the nmap scan.

Minimum Round Trip Time Timeout (`--min_rtt_timeout <milliseconds>`)
As the nmap scan progresses, the minimum round trip time is automatically adjusted based on the actual response times. If the response times at the beginning of the scan are very small, the response time timeout will decrease to match this efficient response time.

If a network is experiencing throughput issues such as dropped packets, it may be necessary to force nmap to maintain a larger timeout value. Setting the `--min_rtt_timeout` value manually will keep the timeout value at a more reasonable setting for these circumstances. This could result in a longer total scan time, so this option should be configured carefully.

The default minimum round trip timeout value is 100 milliseconds. If the minimum is configured to be more than 50,000 milliseconds (50 seconds!), nmap will provide this warning:
```
Warning:  min_rtt_timeout is given in milliseconds, your value seems
pretty large.
```

Maximum Round Trip Time Timeout (`--max_rtt_timeout <milliseconds>`)
The `--max_rtt_timeout` is used to determine when a response is taking too long. If a response takes longer than the configured value, nmap will categorize the request as lost and retransmit or time out the request.

The default maximum round trip time timeout is 10,000 milliseconds (10 seconds). If the nmap scan is operating across slow networks or networks with communication problems, this value may need to be increased to ensure an accurate scan.

The minimum possible `--max_rtt_timeout` value is 5 milliseconds. Although it's possible to set the value this low, it's impractical to run an nmap scan with such a low setting. If a `--max_rtt_timeout` value of less than 20 milliseconds is specified, nmap will provide a warning:
```
WARNING: You specified a round-trip time timeout (10 ms) that
is EXTRAORDINARILY SMALL.  Accuracy may suffer.
```

Parallel Host Scanning

Multiple hosts can be specified on the nmap command line, and nmap's efficient scanning processes will often probe these hosts simultaneously to provide the most efficient scan possible. However, there may be instances when the minimum or maximum number of parallel host scans should be modified.

Maximum Parallel Hosts per Scan (`--max_hostgroup <number>`)
Nmap is extremely efficient when scanning groups of hosts simultaneously. Unfortunately, this efficiency also requires that nmap complete the scanning of a host block before the results can be displayed. At other times, it may be more appropriate to scale the number of simultaneous hosts to a large number to improve the efficiency during a batch scanning process. By adjusting the `--max_hostgroup` option, the number of parallel host scans can be modified to suit any situation.

Many port scans support scanning hosts in parallel, but not all scan types will make use of this option. For example, the ping scan (`-sP`) uses its own grouping algorithm and completely ignores the `--max_hostgroup` option.

The default maximum host group size is 100,000 hosts. This is set intentionally high so that the scan will not be restrictive unless the user manually sets a maximum value.

Minimum Parallel Hosts per Scan (`--min_hostgroup <number>`)
The minimum number of parallel hosts to scan option (`--min_hostgroup`) allows the nmap scan to maintain a high level of efficiency and throughput when scanning a large group of IP addresses in unattended batch processes. This option forces nmap to scan at least this number of hosts in parallel, although these simultaneous scans will require more memory than smaller host groups.

This minimum value must not be larger than the maximum (`--max_hostgroup`), or nmap will provide an error message and cancel the scan. The default number of minimum hosts is one.

If `--min_hostgroup` is configured with more than 100 simultaneous host scans, nmap will provide this warning message:

```
Warning: You specified a highly aggressive --min_hostgroup.
```

This large group may be required, but nmap provides this sanity check to make sure you know what you're getting into.

Parallel Port Scanning
Just as multiple hosts can be scanned in parallel, multiple ports on a single host can also be scanned in parallel. This combination of host and port number is called a socket, and nmap provides complete control over how many sockets can be simultaneously scanned.

Maximum Number of Parallel Port Scans (`--max_parallelism <number>, -M`)
The `--max_parallelism` option specifies the maximum number of ports that can be scanned simultaneously.

The nmap man page describes `--max_parallelism` and `–M` as separate options, but the nmap source code shows that `–M` is simply an alias for `--max_parallelism`.

By default, nmap will simultaneously scan 36 sockets. The `--max_parallelism` option can be set to 1 to scan a single port at a time. The minimum possible value for `--max_parallelism` is 1, and nmap will warn if the requested value is above 900:

```
Warning: Your max_parallelism (-M) option is absurdly high!
Don't complain to Fyodor if all hell breaks loose!
```

You have been warned. :P

Minimum Number of Parallel Port Scans (`--min_parallelism <number>`)
The minimum number of parallel port scans option (`--min_parallelism`) configures nmap to scan at least this number of ports simultaneously. Although this isn't a low-profile option, it does increase the efficiency when scanning firewalled hosts.

This option can overwhelm end stations, and if this option is set greater than 100 then nmap will display this warning:

```
Warning: Your --min_parallelism option is absurdly high!
Don't complain to Fyodor if all hell breaks loose!
```

Delay
Along with the timeout variables, parallel host scanning options, and simultaneous port probes, nmap can also modify the delay between each probe frame to scan as quickly or as slowly as desired. This delay can be modified to come in "under the radar" of an intrusion prevention device, or to slow the scan down for use over non-terrestrial links.

Minimum Delay Between Probes (`--scan_delay <milliseconds>`)

The minimum delay between probes option (`--scan_delay`) takes a departure from the normal nmap nomenclature, since the preface of 'min' isn't included as part of the option name.

The `--scan_delay` option specifies the minimum amount of time that nmap will wait between each port request.

The `--scan_delay` starting value is zero. Nmap dynamically changes this delay during a scan, especially when a remote station (such as FreeBSD or Solaris) begins throttling the responses. When this occurs, nmap displays a message like this:

```
Increasing send delay for 192.168.0.99 from 0 to 5 due to
13 out of 43 dropped probes since last increase.
```

If the `--scan_delay` is specified on the command line, the variable must be greater than zero. If it's set too low, nmap aborts the scan with this nondescript failure message:

```
scan_delay must be greater than 0
```

Maximum Delay Between Probes (`--max_scan_delay <milliseconds>`)

The maximum delay between probes option (`--max_scan_delay`) sets an upper ceiling for the time that nmap will delay between each request.

Nmap's maximum probe delay can dynamically grow to as much as one per second. Using the `--max_scan_delay` option allows for a different maximum delay, perhaps even exceeding the one second maximum default. Increasing this option will significantly slow the total scan time. However, slowing this value may be necessary to obtain accurate scans on non-terrestrial network links or congested WAN connections.

The default maximum delay is 1,000 milliseconds (1 second). If the `--max_scan_delay` is set too low, nmap will halt the scan with this message:

```
max_scan_delay must be greater than 0
```

Timing Policies (`--timing, -T<0|1|2|3|4|5>`)

If nmap's ten different timing options are too much to handle, then the built-in timing policies (`--timing`) may be a good alternative. Timing policies are pre-built groups of timing options that range from the nearly invisible "paranoid" option to the overly-aggressive "insane" category.

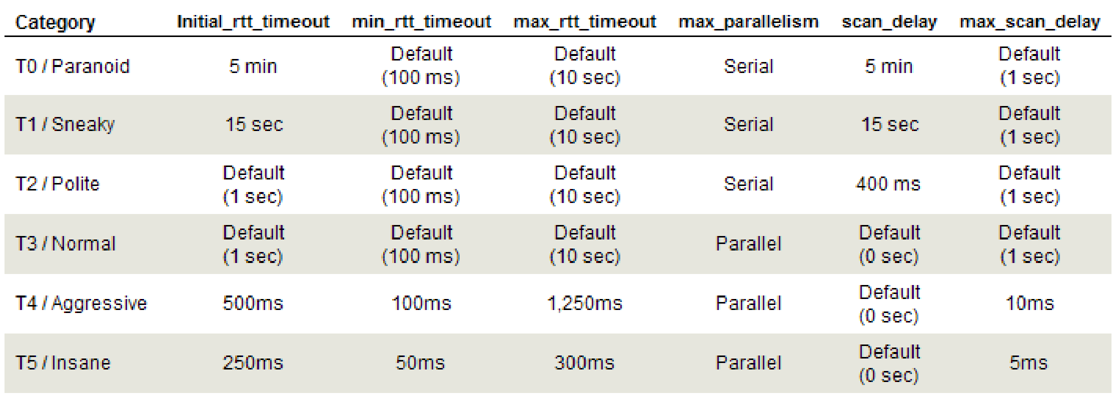
Timing policies are specified as Paranoid, Sneaky, Polite, Normal, Aggressive, and Insane. These categories can also be referenced as numbers, where Paranoid has a value of zero and Insane has a value of five. Therefore, the nmap command

```
# nmap 192.168.0.1 –T0
```

is the same as

```
# nmap 192.168.0.1 --timing paranoid
```

The details of the timing options are shown in this chart. The policies only depart slightly from the defaults, but the departures at the upper and lower ends of the spectrum are significant:



Category	Initial_rtt_timeout	min_rtt_timeout	max_rtt_timeout	max_parallelism	scan_delay	max_scan_delay
T0 / Paranoid	5 min	Default (100 ms)	Default (10 sec)	Serial	5 min	Default (1 sec)
T1 / Sneaky	15 sec	Default (100 ms)	Default (10 sec)	Serial	15 sec	Default (1 sec)
T2 / Polite	Default (1 sec)	Default (100 ms)	Default (10 sec)	Serial	400 ms	Default (1 sec)
T3 / Normal	Default (1 sec)	Default (100 ms)	Default (10 sec)	Parallel	Default (0 sec)	Default (1 sec)
T4 / Aggressive	500ms	100ms	1,250ms	Parallel	Default (0 sec)	10ms
T5 / Insane	250ms	50ms	300ms	Parallel	Default (0 sec)	5ms

As this chart shows, the Paranoid option's minimum `--scan_delay` is set to an amazingly slow 5 minutes, and the Insane option's `--max_scan_delay` is set to 5 milliseconds! Clearly, there are massive differences between the top end of the scale and the bottom end.

If an unknown timing policy is specified, nmap provides this error before halting:

```
Unknown timing mode (-T argment).  Use either "Paranoid",
"Sneaky", "Polite", "Normal", "Aggressive", "Insane" or
a number from 0 (Paranoid) to 5 (Insane)
```

Although the Paranoid category scan provides very accurate information, it takes a very long time to process a single `-T0` scan. Conversely, an Insane scan provides lightning fast results, but the results can be inaccurate if the end station does not respond within the tight timeframes required by the `–T5` category.

These timing categories provide a method of customizing scans based on a standard starting point. Since nmap's timing options are based on their position on the command line, a timing policy can be selected at the beginning of the nmap command line and other individual timing options can be specified afterwards. This would create a customized combination of nmap timings without requiring the specification of every possible timing option on the command line.

These timing policies are useful for testing intrusion detection systems (IDS) and intrusion prevention systems (IPS). Each timing policy can be run against an IDS or IPS to see if an alarm event or packet filtering function occurs. Additional IDS and IPS thresholds can then be configured based on these triggered values.

Chapter 9:

Windows-only Nmap Options
Many Microsoft Windows systems (especially Windows XP Service Pack 2) are hindered with networking restrictions that limit nmap's effectiveness. Even with these technical challenges, nmap still performs admirably in most Windows environments. Because of the large number of Windows-based installations, nmap will undoubtedly be operated from a Windows desktop for the foreseeable future. As these Windows-only options show, there are a number of workarounds and tests that can be used in a Windows environment to optimize nmap's efficiency.

WinPcap and Raw Socket Options
For nmap to work in Windows, it requires the installation of WinPcap 3.14 beta 4 or later. The WinPcap packet capture library is well supported in the open source community, and it has become one of the most popular libraries for Windows-based operating systems.

Nmap's reliance on WinPcap is partly based on the changes that Microsoft has made to the raw sockets functionality on Windows XP Service Pack 2. Microsoft has effectively removed any embedded raw sockets capability in Windows XP SP2, and nmap's architecture changed so that these limitations could be circumvented.

In the past, nmap used Windows built-in raw sockets functions to communicate at the packet level. With the removal of raw sockets, nmap had to find other methods of communicating to the network through Windows. Fortunately, the WinPcap library provides packet-level communication without requiring the Windows XP raw socket functionality.

Other Windows Challenges

Networking in Windows-based operating systems has been a moving target through the years. Early versions of Windows didn't provide any built-in TCP/IP networking, and users that needed a TCP/IP stack were required to use add-on protocol stacks from 3rd party companies. As successive Windows versions integrated additional networking functionality, the internal workings of the Windows operating system changed. Internal networking functions in early versions of Windows were completely rewritten or removed in later versions. This has created challenges for developers, since they must support many different networking techniques to accomplish the same functionality across different Windows versions.

For example, simply gathering a list of the network interfaces on a Windows-based system varies dramatically between different OS versions. Nmap queries the specific Windows version and can react accordingly based on the operating system version in use.

In some cases, nmap even has different methods of obtaining information within the same operating system version.

Help for Windows (`--win_help`)

The `--win_help` option provides an on-screen display of the following Windows-related nmap commands:

```
Windows-specific options:

 --win_list_interfaces : list all network interfaces
 --win_norawsock       : disable raw socket support
 --win_forcerawsock    : try raw sockets even on non-W2K systems
 --win_nopcap          : disable winpcap support
 --win_nt4route        : test nt4 route code
 --win_noiphlpapi      : test response to lack of iphlpapi.dll
 --win_trace           : trace through raw IP initialization
```

The `--win_help` option isn't mentioned in the nmap man page, but it does appear on Windows operating systems in the online help screen (`-h`).

List All Network Interfaces (`--win_list_interfaces`)

Windows doesn't have an easy naming convention for its interfaces, so this options lists the network interfaces with names that nmap recognizes:

```
Available interfaces:

Name        Raw mode  IP
loopback0   SOCK_RAW  127.0.0.1
```

```
eth0        winpcap   192.168.0.5
```

This option is useful when the nmap interface option (`-e`) needs to reference the Windows interface name, or information from nmap's `--debug` output needs to be correlated back to a physical network interface.

Disable Raw Socket Support (`--win_norawsock`)

Some Windows configurations can experience problems running certain nmap scans when raw sockets are used. For example, Windows 2000 with no service pack could blue-screen-of-death (BSoD) if an IP protocol scan (`-sO`) was requested. Fortunately, nmap already identifies a Windows 2000 system with this exposure and disables raw sockets automatically if an IP protocol scan is selected.

The disable raw socket support option (`--win_norawsock`) will administratively disable all raw socket support for the duration of the nmap scan. This option is probably not necessary in Windows NT and Windows 98, since those operating systems never supported raw sockets.

This option can help to get around raw sockets problems in any Windows version. If this option is used, nmap will default to using WinPcap as the communications mechanism.

Try Raw Sockets Even on non-W2K Systems (`--win_forcerawsock`)

Nmap will automatically identify the Windows operating system on which it runs, and it will make changes to its communications methods based on these differences in Windows versions. In the cases where nmap has automatically disabled raw sockets, it may be helpful to force the use of raw sockets for troubleshooting purposes.

Forcing nmap to use raw sockets doesn't necessarily mean that Windows will use raw sockets exclusively. For example, there are some Windows XP-related nmap functions that will only work properly with WinPcap. In these situations, both raw sockets and WinPcap are available and nmap can choose which function to use. If WinPcap should be disabled, the `--win_nopcap` option should be used.

Disable Winpcap Support (`--win_nopcap`)

The `--win_nopcap` option disables nmap support for the WinPcap library. Instead of using the normal WinPcap library calls, nmap attempts to use raw sockets.

If this option is enabled, the following line appears in the `--win_trace` output:

```
***WinIP***  winpcap support disabled
```

If both `--win_nopcap` and `--win_norawsock` is used, nmap is left with no communication mechanism and the following "strange" error message is displayed:

```
EXC selected for machine 192.168.0.99
Strange read error from 192.168.0.99: Unknown error
```

Enabling this option on my Windows XP SP2 system prevented the nmap pings from working, although the actual scan process appeared to work correctly.

Test NT 4.0 Route Code (`--win_nt4route`)
For nmap to send packets, it must identify the physical network interfaces and decide which connection to the remote device is the best. However, the changes to Windows versions through the years have created many different methods of obtaining this information.

The `--win_nt4route` option uses an alternative method of finding the best route for nmap to use. If nmap is having difficulty locating a route to a remote device, this option may provide a workaround.

When using this option, the following text can be seen in a level-two debug output (`-d -d`):

```
get_best_route: using NT4-compatible method
```

Test Response to Lack of iphlpapi.dll (`--win_noiphlpapi`)
The `--win_noiphlpapi` refers to the use of the `iphlpapi.dll` library. This library is used by nmap functions to identify network interfaces and send ARP requests.

If this option is selected, nmap will use alternative options for obtaining this IP-specific information. These non-iphlpapi functions are also used if a Windows version doesn't support the IP helper library, such as Windows 95. The IP helper library is already included with the most recent Windows operating systems.

If nmap cannot find the IP helper library, this following message will be displayed:

```
Your system doesn't have iphlpapi.dll

If you have Win95, maybe you could grab it from a Win98 system
If you have NT4, you need service pack 4 or higher
If you have NT3.51, try grabbing it from an NT4 system
Otherwise, your system has problems ;-)
```

Trace Through Raw IP Initialization (`--win_trace`)
The `--win_trace` option is extremely useful for Windows users who are having problems running nmap. The Windows IP initialization process includes the identification of the interfaces, the identification of communication libraries (WinPcap and raw sockets), and the recognition of a privileged Window user.

The Windows IP trace option displays this information with the normal nmap output:

```
***WinIP***  initializing if tables
***WinIP***  if tables complete :)
***WinIP***  trying to initialize winpcap 2.1
***WinIP***  winpcap present, dynamic linked to: WinPcap version 3.1
beta4 (pack
et.dll version 3, 1, 0, 24), based on libpcap version 0.8.3
***WinIP***  testing for raw sockets
***WinIP***  rawsock is available
***WinIP***  reading winpcap interface list
***WinIP***  init \Device\NPF_GenericNdisWanAdapter (ASCII)
pcap device:  \Device\NPF_GenericNdisWanAdapter
***WinIP***  init \Device\NPF_{D37FB73B-773F-4941-B09E-51D487A0B724}
(ASCII)
pcap device:  \Device\NPF_{D37FB73B-773F-4941-B09E-51D487A0B724}
```

```
 result:       physaddr (0x00114343a834) matches eth0
***WinIP***  o.isr00t = 1
```

Skip IP Initialization (`--win_skip_winip_init`)

The `--win_skip_winip_init` option completely bypasses the Windows-related IP initialization process. This option is useful during a troubleshooting session where the Windows IP initialization process is in question.

If nmap starts normally and begins the scan without initializing the Windows IP system, the following message is displayed:

```
winip not initialized yet

QUITTING!
```

Chapter 10:

Miscellaneous Options

There are a number of nmap options that don't fall into any of the previous categories. These options provide functions such as version information, custom TCP flag creation, and argument vector quashing.

Quick Reference Screen (`--help, -h`)

The `--help` option provides a summary of commands, including at least one option, `--interactive`, that is not mentioned in the nmap man page:

```
Nmap 3.81 Usage: nmap [Scan Type(s)] [Options] <host or net list>
Some Common Scan Types ('*' options require root privileges)
* -sS TCP SYN stealth port scan (default if privileged (root))
  -sT TCP connect() port scan (default for unprivileged users)
* -sU UDP port scan
  -sP ping scan (Find any reachable machines)
* -sF,-sX,-sN Stealth FIN, Xmas, or Null scan (experts only)
  -sV Version scan probes open ports determining service & app
names/versions
  -sR RPC scan (use with other scan types)
Some Common Options (none are required, most can be combined):
* -O Use TCP/IP fingerprinting to guess remote operating system
  -p <range> ports to scan.  Example range: 1-1024,1080,6666,31337
  -F Only scans ports listed in nmap-services
  -v Verbose. Its use is recommended.  Use twice for greater effect.
  -P0 Don't ping hosts (needed to scan www.microsoft.com and others)
* -Ddecoy_host1,decoy2[,...] Hide scan using many decoys
  -6 scans via IPv6 rather than IPv4
  -T <Paranoid|Sneaky|Polite|Normal|Aggressive|Insane> General timing
policy
  -n/-R Never do DNS resolution/Always resolve [default: sometimes
resolve]
  -oN/-oX/-oG <logfile> Output normal/XML/grepable scan logs to
<logfile>
  -iL <inputfile> Get targets from file; Use '-' for stdin
* -S <your_IP>/-e <devicename> Specify source address or network
interface
```

```
  --interactive Go into interactive mode (then press h for help)
  --win_help Windows-specific features
Example: nmap -v -sS -O www.my.com 192.168.0.0/16 '192.88-90.*.*'
SEE THE MAN PAGE FOR MANY MORE OPTIONS, DESCRIPTIONS, AND EXAMPLES
```

On Windows-based systems, the `--win_help` option is also displayed in this list.

Nmap Version (`--version, -V`)

The nmap version option (`--version`) simply outputs the nmap version number and URL:

```
nmap version 3.81 ( http://www.insecure.org/nmap )
```

This can be useful if multiple or unknown nmap versions are installed on a system and the exact version needs to be identified.

Data Directory (`--datadir <directory_name>`)

The data directory option (`--datadir`) administratively configures the directory location containing the nmap support files. For more information on the `--datadir` option and the nmap support files, refer to Nmap Support Files in Section I.

Quash Argument Vector (`-q`)

With the quash argument vector option (`-q`), nmap can modify its argument vector to appear as if it's another process. By default, the `–q` option causes nmap to appear as the Program for Internet News & Email, or "pine." Since pine is a common email package, this option allows nmap to blend with other system processes:

```
# nmap 192.168.0.1 –q &
# ps r

  PID TTY              STAT   TIME COMMAND
 3772 /UNIONFS/dev/pts/1 R+   4:03 pine & the "quashed" nmap session
 4056 /UNIONFS/dev/pts/2 R+   0:00 ps r
```

Define Custom Scan Flags (`--scanflags <flagval>`)

The `--scanflags` option provides the nmap user with a one-size-fits-all method of scanning the network. Using `--scanflags` provides the user with a method of controlling the TCP flags used during the nmap scan. With this option, 256 different TCP flag settings can be used to create a unique scanning experience!

The flag values can be specified with a decimal representation of the TCP flag byte, a hexadecimal value (preceded by 0x), or the name of the flag. Valid flag names are URG, ACK, PSH or PUSH, RST or RESET, SYN, and FIN. An nmap command line of:

```
# nmap 192.168.0.99 --scanflags URGACKPUSHRESETSYNFIN
```

is the same as

```
# nmap 192.168.0.99 --scanflags 63
```

and is the same as

```
# nmap 192.168.0.99 --scanflags 0x3F
```

The output from this nmap command line shows this information in the TCP header:

```
TCP: ----- TCP header -----
      TCP:
      TCP: Source port              = 36797
      TCP: Destination port         =   443 (Https)
      TCP: Initial sequence number = 3256272004
      TCP: Next expected Seq number= 3256272005
      TCP: Acknowledgment number   = 0
      TCP: Data offset             = 20 bytes
      TCP: Reserved Bits: Reserved for Future Use (Not shown in the Hex
Dump)
      TCP: Flags                   = 3F
      TCP:               ..1. .... = Urgent pointer
      TCP:               ...1 .... = Acknowledgment
      TCP:               .... 1... = Push
      TCP:               .... .1.. = Reset
      TCP:               .... ..1. = SYN
      TCP:               .... ...1 = FIN
      TCP: Window                  = 2048
      TCP: Checksum                = 0DD7 (correct)
      TCP: Urgent pointer          = 0
      TCP: No TCP options
      TCP:
```

Although this option functions properly, it has limited use unless there are specific circumstances that might require an additional scan method.

(Uriel) Maimon Scan (`-sM`)

The Maimon scan (`-sM`) is similar to the FIN scan (`-sF`), but the Maimon scan enables both the ACK flag and the FIN flag in the frame sent to the remote device. This stealth scan is designed to operate much like the FIN scan. If a RST is received, the port is closed. If no response is received, the port is open. The Maimon scan is named after Uriel Maimon, who profiled stealth scanning techniques in Phrack 49, Article 15.

Like the FIN scan, the Maimon (FIN/ACK) scan is designed to identify RST responses from the remote device as closed ports, and no response as an open port. However, this isn't how this FIN/ACK combination tends to work in the real world. Most of the devices tested in the NetworkUptime.com lab interpreted the probes as ACK packets, as if the FIN didn't exist. Receiving a TCP reset was actually identification of an open port, not a closed port. The final scan report will still show the ports that have been found, but it will identify them as 'closed.'

The Maimon scan is also very dependent on the operating systems in use. Windows XP SP2 wasn't susceptible to the Maimon scan, but Windows XP without a service pack readily identified open ports (although nmap improperly labeled them as closed).

This limited functionality positions the Maimon scan as the only nmap scan that is not mentioned in the man page or in the online help (`-h`). The Maimon scan is referenced in the changelog and the nmap source code, and it runs as expected from the nmap command line.

IPv6 Support (`-6`)
Nmap includes support of IP version 6 (IPv6) devices with the `-6` option. The end stations must be addressable via IPv6, even if the nmap station is not. IPv6 addresses are expressed as eight groups containing two hexadecimal numbers in each group, and the groups are usually separated by colons. An IPv6 address would look something like this:

```
50AF:1011:10EF:5504:38D0:1211:0000:747C
```

The IPv6 support in nmap is currently limited to the connect() scan (`-sT`), the ping scan (`-sP`), and the list scan (`-sL`).

CHAPTER 11:

USING NMAP IN THE "REAL WORLD"

Now that all of these nmap scan types, option settings, and information screens have been documented, how can this conglomeration of data help an organization with real-world security requirements?

The following scenarios are common examples that are found in many organizations. These are written from the perspective of a network or security manager who provides uptime and availability of an organization's systems. It's also assumed that the security team will have nmap scanning systems that run with privileged access.

With every nmap scan, there are some options that are always recommended. The verbose option should always be specified at its highest level (`-vv`), and the universal output format (`-oA`) should also be used. Since the differently formatted nmap output files occasionally contain different pieces of information, saving the nmap information into all formats can provide important information after the scan. It's also assumed that the security team will have nmap scanning system that run with privileged access.

The `--excludefile` option should also be used with every nmap scan. The exclude file should be updated with the most important IP addresses in the organization, or the IP addresses that should never be scanned under any circumstances. Nmap scans that operate without incident on some systems may have far-reaching effects on others! For example, an older telephone system or a legacy router may not be able to provide the resources required by nmap, and these systems may crash or become unavailable if the nmap scan is too aggressive. Exclude options always take priority over any includes, so identifying these IP addresses in an exclude file will ensure that they are never scanned by nmap.

Identifying the Remnants of a Virus Outbreak or Spyware Infestation

Viruses and spyware have different underlying technologies, but they have a common bond once they infest a system. Variants of MyDoom, Sasser, Beagle, NetBus, SubSeven, and other Trojan horses create open ports, providing backdoor communication conduits into infected systems.

This scan will show how the entire network can be easily scanned to locate a single spyware or virus remnant. This search will make use of unique ping methods, port searches, and reverse DNS lookup settings.

Identifying Virus, Spyware, and Trojan Horse Remnants

- **Nmap Ping type:** If ICMP is unfiltered in an organization's network, an ICMP ping (`-PE`) would be an efficient way of identifying an active system. If ICMP is actively filtered, a more applicable ping type should be considered. Since this scan will be working through a large number of IP addresses, an nmap ping will be an important method of determining a remote device's availability. Disabling nmap's ping option (`-P0`) should be used only as a last resort in a scan with a large number of IP addresses.

- **Nmap Scan Type:** Since this is a simple port availability test, a TCP SYN scan (`-sS`) or a UDP scan (`-sU`) would be an effective scan type. Different spyware remnants can open TCP ports or UDP ports, and occasionally both types will need to be specified. This example will assume that both types will be scanned.

- **IP Addresses:** The IP addresses for these types of scans will usually be a range of addresses, using nmap's wildcards and naming conventions. For ease of use, these IP addresses should be listed in a file that can be included with the `–iL`option.

- **Port Ranges:** This particular scan will only need to scan a few ports from each device that are associated with a specific spyware infestation. If both TCP and UDP scan types are specified, then the `–p` option should include `U:<udpports>,T:<tcpports>`.

- **Reverse DNS:** If a large number of hosts will be scanned over a long timeframe, it may be helpful to require a reverse DNS on each scanned device (`-R`). However, this can create delays during the scan process, and it may not be required if the IP addresses on the network rarely change. In these cases, a more efficient scan would disable the reverse DNS resolution process (`-n`).

- **Version Detection:** If a Trojan horse is using a port number that is commonly open on the network, it may be helpful to include a version detection option (`-sV`) to help identify the application type running on the remote device. This will slow down the scan significantly (especially if UDP scanning is active), so use this option only if necessary.

- **Time to Live:** If this scan shouldn't traverse a WAN or slow network link, the `--ttl` option may be useful. This would prevent the scan from using slower WAN links, although it may be more reliable to add the remote IP addresses to the exclude list if the exact network configuration isn't known.

The nmap command line would be similar to this (disregard the line break):
```
# nmap –vv –PE –sS –sU –iL input.lst --excludefile banned.lst
  –p U:31337,T:6713 –n -oA trojans
```
After the scan is complete, the grepable output file (`trojans.gnmap`) can be searched for the word "open" to determine if any open ports were identified.

Vulnerability Assessments
Vulnerability announcements are a daily occurrence in the security world. Servers, applications, routers, switches, desktop computers, any other network connected device has the potential to be exploited. When problems are identified with a device or application, announcements are usually made through security mailing lists and manufacturer web pages.

When the public vulnerability announcement is made, the problem has been identified, testing has been completed, and the manufacturer has created a patch for the problematic operating system, application, or driver. If the vulnerability affects many systems or the problem has dramatic security repercussions, it's important to patch the affected systems as quickly as possible.

When an application must be upgraded, the scope of the upgrade may not be readily apparent. The security team may know of most systems using the vulnerable software, but there may be other systems on the network of which the security team is unaware. This was a common problem in January of 2003 when the SQL Slammer worm attacked Microsoft SQL Server systems. Although many network teams had patched all known SQL Server systems, many organizations had non-production SQL Server systems that were not known. These unknown and unpatched systems were quickly infected and the vast flood of network traffic created by the SQL Slammer worm creative massive network disruptions.

The nmap scan can locate application services that are using known port numbers, and nmap's version scan can provide more information about the application service. Nmap's customized application fingerprints can even provide the application version number, in some cases.

Vulnerability Assessment Details

- **Nmap ping type:** If ICMP is not filtered, an effective nmap ping is the ICMP echo request ping (`-PE`). The nmap ping requirements are important, since the entire network will be scanned.

- **Nmap scan type:** In many cases, a limited number of ports would be scanned. If Microsoft SQL Server was the concern, then UDP port 1434 would be scanned. Since nmap only has a single UDP scan (`-sU`), the scanning option in this example is limited to this single scan type. If the destination port was TCP-based, many different scan types could be selected based on speed and accuracy.

- **IP Addresses:** The IP addresses will usually be a range of addresses that covers the entire network. Nmap's wildcards and naming conventions can be used to specify subnets and address ranges. In most cases, these IP addresses will be saved in a file that can be included with the `–iL` option.

- **Port Ranges:** Microsoft SQL Server's monitor access is through UDP port 1434. In this example, this UDP port is the only port that needs to be scanned. Specifying the ports with a `U:` or `T:` specification is unnecessary, but it can often be helpful for clarification.

- **Reverse DNS:** In an organization with many hosts, a reverse DNS may assist by identifying a remote device by name. This name resolution process will add delays to the scan, but the identification of the host name may outweigh the time delay associated with the DNS lookups.

- **Version Detection:** The version detection option (`-sV`) is the key to this scan. In many cases (such as this one), nmap can provide version information based on the included version signatures.

This SQL Server scan would be based on this nmap command line (disregard the line break):

```
# nmap -vv -PE -sU –iL input.lst –-excludefile banned.lst
  -p U:1434 -n -oA sql_svr -sV
```

The nmap output shows the identification of the Microsoft SQL Server monitor port and detailed version information:

```
Interesting ports on 192.168.0.3:
PORT     STATE SERVICE  VERSION
1434/udp open  ms-sql-m Microsoft SQL Server 8.00.194 (ServerName: DT;
TCPPort: 1433)
MAC Address: 00:30:48:27:2C:2A (Supermicro Computer)
```

The version scan provides the SQL Server version number, the server name, and the TCP port used by inbound remote connections to the SQL Server application. Notice that this is a different port than the one used in the nmap scan.

With this information, the network administrators can clearly identify all Microsoft SQL

Server processes, the IP addresses for all SQL Server devices, and the version number information.

Security Policy Compliance Testing
An important aspect of network security is the control and administration of applications and devices into the organization's infrastructure. The security administrator may decide that certain applications pose a risk to the organization's data, and those particular services would be forbidden from attaching to the network.

For example, an organization may decide that the Apache HTTP Server is appropriate for use on the network, but the Microsoft Internet Information Server is not. Since both of these services use TCP port 80, a simple port scan would not provide enough information to determine if the remote device was compliant with the security policies.

Fortunately, nmap can determine much more information than which ports are open or closed. Nmap's version scan can be invaluable for providing detailed information about the active services on a remote device. Since compliance testing also includes operating system information, this example will include nmap's operating system fingerprinting option.

Security Policy Compliance Testing Details

- **Nmap Ping Type:** The nmap ping type will be especially important with this scan type, since an intentionally noncompliant device may have enabled firewall or privacy functions that would hide it from the rest of the network. Nmap's ping options can be combined together to maximize the possibility of finding a device on the network. In this example, the default ACK ping on port 80 (`-PA80`) and ICMP ping (`–PE`) will be used, as well as a SYN ping on port 23 (`–PS23`) for good measure.

- **Nmap Scan Type:** For compliance testing that focuses on web services, a TCP SYN scan (`-sS`) should be sufficient to determine the available ports on a remote device. If the compliance testing is concerned with specific operating systems, the stealth scans (`-sF, -sX, -sN, -sM`) or the window scan (`-sW`) may be an applicable alternative.

- **Operating System Fingerprinting:** If the security policy includes operating system standardization, the OS fingerprinting option (`-O`) should be included. If the operating system type isn't important, this option can be ignored. If the network is large and many systems will be scanned, the `--osscan_limit` option can provide faster and more accurate operating system fingerprints.

- **Version Detection:** Once nmap identifies open ports, it can also query each open port to identify the application and version information associated with the port. If both the version detection option and the OS fingerprinting option are required, the nmap additional, advanced, and aggressive option (`-A`) can be used as a shortcut.

- **Port Ranges:** Since many ports will be scanned across many different devices, it may be more applicable to use nmap's fast scan option (`-F`). If there are certain ports that are important in the compliance check, they should be added to the `nmap-services` support file (if they aren't in there already). This should also increase the overall speed of the scan, since the number of scanned ports will be limited to those in the `nmap-services` file.

- **Reverse DNS:** If an out-of-compliance device is identified, the name of the device may be important. If the organization's DNS is integrated into the remote station IP addressing conventions, then nmap should always perform a reverse DNS (`-R`). If the IP addresses and names are not linked, then nmap will be the most efficient if it never performs a reverse DNS lookup (`-n`).

The nmap security policy compliance scan should look something like this (disregard the line break):

```
# nmap -vv –PA80 –PE –PS23 –sS –iL input.lst --excludefile banned.lst
  -A --osscan_limit –F –R -oA policy_check
```

This sample of the output shows ports, operating systems, and version information:

```
Interesting ports on 192.168.0.5:
(The 1217 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE      VERSION
135/tcp  open  msrpc        Microsoft Windows msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds Microsoft Windows XP microsoft-ds
5101/tcp open  admdog?
MAC Address: 00:0F:B5:0A:6E:DA (Netgear)
Device type: general purpose
Running: Microsoft Windows 2003/.NET|NT/2K/XP
OS details: Microsoft Windows 2003 Server or XP SP2
```

The grepable output file can be searched for details, or the entire scan can be viewed in a web browser with the XML file.

Asset Management

As networks become more distributed, managing remote assets becomes a challenge. A remote location may not be large enough to have its own security staff on site, but the systems and processes at the remote location are still an important part of the organization's overall mission. If there are many remote sites or the remote locations are growing, it will become increasingly difficult to know exactly what devices and systems are installed in the organization. This also becomes important if software licensing is handled by number of "nodes" (common for anti-virus software licensing), or if server-based software is licensed on a per-server basis.

If a simple "show of hands" is required for a node-based licensing agreement, nmap can locate, identify, and categorize the remote devices on the network. These scans can be as detailed or as simple as necessary. This example will show how nmap can identify remote devices across a slow WAN link without affecting normal production traffic.

Asset Management Details

- **Nmap Ping Type:** For this simple node count, the default nmap ping will be sufficient. If ICMP filtering or packet filtering is enabled between the nmap station and the remote devices, a different ping type may be required to circumvent the filtering.

- **Nmap Scan Type:** To determine how many nodes exist at a remote location, nmap only needs to ping the devices on the input list (`-sP`). If operating system fingerprinting is required, a ping scan cannot be used. If an operating system fingerprint is required, a SYN scan (`-sS`) would most likely provide the best overall TCP information.

- **Operating System Fingerprinting:** OS fingerprinting (`-O`) uses very few packets to determine what kind of system exists on the other end of the network. Since the nmap timing policy (see below) will throttle the communications flow, the fingerprinting process will not overwhelm the slow WAN link or affect the normal flow of network traffic. The OS fingerprinting process is also helpful if the asset inventory requires that routers, switches, and printers are categorized separately from the client and server systems. In this example, OS fingerprinting will not be required.

- **Reverse DNS:** The reverse DNS lookup is optional. This asset management task won't require the association of IP addresses to individual names (`-n`).

- **Timing Policy:** The nmap timing policy is one of the most important options of this scan. Since the remote devices are located across a slow WAN link, the nmap processes will need to run slower than normal to avoid any conflicts with production network traffic. The nmap timing policies can adjust nmap's communication processes to be as passive or as aggressive as necessary. In this example, the more passive "polite" policy will be used (`-T Polite`).

The nmap asset management scan will look something like this:

```
# nmap -vv –sS –iL input.lst --excludefile banned.lst -O –n –T Polite -
oA remote
```

The `input.lst` file should include the IP address range of the networks at the remote location. If another remote location needs to be scanned, these additional IP addresses can be included in the input file or a separate file can be created for use in a different nmap scan session.

The "polite" timing policy will run relatively slowly compared to normal nmap scans. This timing can be tweaked further with detailed timing options, but the trade-off to a faster scan will be higher utilization of the WAN link.

Firewall Auditing

Complex firewall rules and configurations create additional challenges for the security team. Does the firewall allow traffic to flow normally to the expected IP addresses? Which ports are open, and which are filtered? To answer these questions, this example will perform a firewall port analysis to show which ports appear open to the outside world.

At its most basic level, a firewall needs to filter or pass network traffic. Firewalls can range from simple packet filters to complex proxy servers. Some firewalls pass IP traffic without modification, and others provide a translation between internal IP addresses and a single external address.

The nmap ACK scan can determine if a packet is being filtered by the firewall or if the packet is passing through unfiltered. The ACK scan doesn't identify open ports, but it can help determine if frames are being filtered.

Firewall Auditing Details

- **Nmap Ping Type:** Most firewalls should have ICMP filtering enabled, so only half of the normal nmap ping will be successful. Nmap's ICMP ping should fail, but the default ACK ping (`-PA80`) will often be successful. Since this scan is usually directed towards a known IP address, it's not necessary to ping the remote device at all (`-P0`)!

- **Nmap Scan Type:** The TCP ACK scan (`-sA`) can describe a filtered or unfiltered port to a remote device without connecting to an application session. This scan type is useful for a firewall audit, although a TCP connect scan (`-sT`) may be required if the firewall is stateful and relies on an active session to pass network traffic.

- **Don't Randomize Ports:** During this audit, it's recommended that a protocol analyzer capture the nmap traffic. If a port is found to be unfiltered, the trace file can confirm the traffic flow and provide a comparison if changes are made to the firewall rules. To make it easier to follow the traffic flow in the protocol decode, the nmap scan should be configured to scan the remote device ports in numerical order (`-r`), and not the default random order.

- **Reverse DNS:** Since the nmap scan will be run on a well-known host, it will not be necessary to do a reverse DNS resolution (`-n`) on the remote device.

- **Fragmented Frames:** Some additional firewall testing might include the fragmentation of frames (`-f, -ff`) to test the ability of the packet filter to allow or filter fragmented traffic.

The firewall audit nmap command line would look like this:

```
# nmap -vv –P0 –sA –iL input.lst --excludefile banned.lst –r -n -oA
firewall
```

Once nmap identifies the unfiltered ports, the firewall may be modified to "tighten" or "loosen" the available servers or services. After each modification, the nmap audit can be used to determine if the expected changes were successful. This audit can also be run periodically to determine if the firewall configuration has been modified since the last audit.

Perpetual Network Auditing

An organization's network is constantly changing. New application servers are added, switch configurations are updated, and traffic flows change as the organization changes. The security team is constantly challenged to understand the interactions of the network at every level.

One of the best ways to understand this "living" network is constant examination. One common strategy is to blanket the network with a perpetual scan in an effort of constant vigilance. Although this strategy isn't all-encompassing, it certainly fits into the concept of defense in depth or layered security. The security team never knows when the information gathered in the past will be valuable in the present.

Perpetual Network Auditing Details

- **Nmap Ping Type:** The default nmap ping of an ICMP ping combined with a TCP ACK on port 80 is usually sufficient when the scan occurs on the local network. If the scan must traverse a firewall or packet filter, the ping should be modified accordingly.

- **Nmap Scan Type:** For passive scanning, the TCP SYN scan (`-sS`) is appropriate. The SYN scan never initiates an application session, and the interaction between the nmap station and the remote device is only at the TCP transport layer. This scan type has a very low possibility of disrupting remote devices, especially when the network interactivity is administratively reduced through nmap's built-in timing policies.

- **IP Addresses:** Since the range of IP addresses will be randomized (see below), it may be most effective to enter a large portion of the network's IP address ranges into the input file. If the scan is focused on a particular location or IP subnet, the IP address input file can be customized to fit this smaller group of stations.

- **Operating System Fingerprinting:** The goal of this perpetual scan is to obtain as much information as possible with relatively little network interaction. The operating system fingerprinting option (`-O`) only sends fourteen additional frames, but the information received from the scan is extremely valuable. Although some additional accuracy could be garnered by including the `--osscan_limit` option, the goal of this perpetual information gathering process is to obtain the largest quantity of information possible. If this information is required at a later date, the accuracy of the information can be determined at that time.

- **Timing Policy:** This perpetual scan doesn't need to complete in any particular timeframe. This scan should remain as invisible as possible and the availability of the devices on the network should be the primary concern. Setting a timing policy to "polite" (`-T2`) should allow the scan to run at a reasonable pace while maintaining a low amount of interactivity.

- **Reverse DNS:** The assigned IP addresses in dynamic environments may change every day. If the DNS is integrated into the dynamic IP addressing environment, nmap should always perform a reverse name resolution (`-R`).

- **Randomize Hosts:** Since this scan is constantly watching the network, it may be more interesting to have the scan randomly roam from one device to another. The `--randomize_hosts` option will shuffle the list of hosts so that the network interaction is never focused on one particular subnet.

One option that isn't included in the perpetual scan strategy is the version detection option. Unlike the operating system fingerprinting function, version detection is a more invasive method of querying a remote device. Version detection opens application sessions on the remote device, and the version detection information gathering process can consist of many frames for each open port. In an effort to keep this scan passive, simple, and relatively fast, the version scan option has not been included.

The nmap command line for the perpetual scan would look something like this (disregard the line break):

```
# nmap -vv –sS –iL input.lst --excludefile banned.lst
  -O –T2 –R --randomize_hosts -oA perpetual
```

The output files will provide an easy-to-search grepable format, and a corresponding XML format for formal reporting purposes.